NPS52-89-004

NAVAL POSTGRADUATE SCHOOL

Monterey, California

MEANINGFUL REAL-TIME GRAPHICS WORKSTATION PERFORMANCE MEASUREMENTS

MARK A. FICHTEN

DAVID H. JENNINGS

MICHAEL J. ZYDA

NOVEMBER 1988

Prepared for:
US Army Combatt Developments Experimentation Center
Ord, CA. 93941

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral R. C. Austin
Superintendent

Harrison Shull
Provost

This report was prepared in conjunction with research conducted for the United States Army Combat Developments Experimentation Center, the Naval Ocean Systems Center, and the Naval Underwater Systems Center. The work was funded by the Naval Postgraduate School and the United States Army Combat Developments Experimentation Center .

This report was prepared by:

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) NPS52-89-004	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION NAVAL POSTGRADUATE SCHOOL 6b. OFFICE SYMBOL (*If applicable*)	7a. NAME OF MONITORING ORGANIZATION US Army Combat Developments Experimentation Center.
6c. ADDRESS (*City, State, and ZIP Code*) MONTEREY, CA 93943	7b. ADDRESS (*City, State, and ZIP Code*) Fort Ord, CA 93941
8a. NAME OF FUNDING / SPONSORING ORGANIZATION NAVAL POSTGRADUATE SCHOOL 8b. OFFICE SYMBOL (*If applicable*)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER 02MN, Direct Funds and ATEC 44-87, Fort Ord. CA. 93941
8c. ADDRESS (*City, State, and ZIP Code*) MONTEREY, CA 93943	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (*Include Security Classification*)
MEANINGFUL REAL-TIME GRAPHICS WORKSTATION PERFORMANCE MEASUREMENTS

12. PERSONAL AUTHOR(S)
MARK A. FICHTEN DAVID H. JENNINGS, MICHAEL J. ZYDA

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (*Year, Month, Day*)	15. PAGE COUNT
SUMMARY	FROM 1 Apr 88 TO 31Dec88	1988 November	135

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)
FIELD	GROUP	SUB-GROUP	3D Visual simulation systems, graphics workstation performance measurements.

19. ABSTRACT (*Continue on reverse if necessary and identify by block number*)

We present how graphics workstation performance is currently measured and how performance should be measured in the future. Four levels of graphics system performance measurements are low-level primitives (points and Lines), pictures (collections of points and lines), systems (collections of pictures), and applications (collections of systems). The different techniques for measuring performance vary widely depending on the hardware manufacturer, the software programmer, or the article author. This paperdiscusses performance. measurements of real-time graphics application with emphasis on expressing the measurements in common terms.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Michael J. Zyda	22b. TELEPHONE (*Include Area Code*) (408) 646-2305 22c. OFFICE SYMBOL

Meaningful Real-Time Graphics Workstation Performance Measurements[1]

Mark A. Fichten, David H. Jennings and Michael J. Zyda*
Naval Postgraduate School
Code 52, Dept. of Computer Science,
Monterey, California 93943-5100

ABSTRACT

We present how graphics workstation performance is currently measured and how performance should be measured in the future. Four levels of graphics system performance measurements are low-level primitives (points and lines), pictures (collections of points and lines), systems (collections of pictures), and applications (collections of systems). The different techniques for measuring performance vary widely depending on the hardware manufacturer, the software programmer, or the article author. This paper discusses performance measurements of a real-time graphics application with emphasis on expressing the measurements in common terms.

---

[1]This work was supported by the US Army Combat Developments Experimentation Center, Fort Ord, California and the Naval Postgraduate School's Direct Funding Program. This work was generated from Mark A. Fichten's and David H. Jennings' joint Masters Thesis.

*Contact author.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. GRAPHICS SYSTEMS OVERVIEW

A. INTRODUCTION

Graphics workstation performance measurement is a complex and diverse topic. There is currently no standard that is utilized throughout industry or academia to measure the performance of graphics workstations. As a consequence of this, graphics workstation manufacturers quite often choose numbers better than their competitors and then construct programs that match the numbers. Some of the current methods of measuring graphics workstation performance include vectors per second, filled and Z-buffered polygons per second, and pixels per second. The problem with these measurements is that the accessibility of these numbers varies between applications. No standard, exchangeable programs exist to compare the graphics capabilities of different systems. The programs the manufacturers cite are stamped *proprietary* and are generally unavailable.

For graphics workstations, there are four levels of possible performance evaluation. These levels are low-level primitives (points and lines), pictures, systems, and applications [Ref. 1:p. 348]. We believe that applications level performance evaluation is the best overall indicator of a workstation's graphics capabilities. We feel this is true as a viewer can see such a system running and can be convinced that it is either fast enough, too slow, or just right for his application of similar complexity.

For applications level performance evaluation, there are many potential applications from which to choose. For this study, we explore graphics workstation performance evaluation using the visual simulator paradigm. We examine the Moving Platform Simulator (MPS), a simulator of known complexity and operating characteristics, and one whose source is readily attainable. We chose the visual simulator paradigm for performance measurements because this type of system has a realistic mix of CPU and graphics computational requirements. MPS is a real-time, three

dimensional simulation of moving platforms (jeeps, trucks, tanks and missiles) as they maneuver over digital terrain. The MPS system was written entirely by students associated with the Graphics and Video Laboratory of the Department of Computer Science at the Naval Postgraduate School. The goal of our work in the Graphics and Video Laboratory is to develop as accurately as possible real-time three dimensional graphics simulations within the constraints of commercially available graphics hardware. The pictures need to have enough detail to accomplish the selected mission while maintaining a frame update rate which is as fast as possible.

B. HARDWARE SPECIFICATIONS AND PERFORMANCE ESTIMATES

Several manufacturers produce high performance graphics workstations. These include Silicon Graphics Inc. of Mountain View, California [Ref. 2:pp. 239-246], Ardent Computer Corporation of San Jose, California [Ref. 3], and Stellar Computer Inc. of Newton, Massachusetts [Ref. 4:pp. 255-256]. Table 1.1 compares the manufacturers' rated performance of their workstations.

The numbers cited in Table 1.1 are meaningless as accurate indicators of relative performance. We can make such a statement as we do not have **complete** information as to how these numbers were derived. We do not know what techniques were utilized nor do we even have the programs from which these numbers were generated. We can get a more accurate measurement of the performance of such workstations if we instead utilize them for an application of known complexity for which the source is available.

TABLE 1.1 MANUFACTURERS' SPECIFICATIONS

WORKSTATION	CPU	CAPABILITY	NOTES
IRIS 3120	2 MIPS	1000 10x10 pixel polygons	1
IRIS 4D/70G	10 MIPS	5500 polygons	1
IRIS 4D/70GT	10 MIPS	60,000 polygons	1
		120,000 triangles	
		40,000 10x10 pixel quadrilaterals	2
Stellar GS1000	25 MIPS	150,000 100 pixel triangles	3
Ardent Titan	16 MIPS (4)	200,000 triangles	1
		400,000 vectors	

Notes:
1 - Z-buffered, Gouraud shaded
2 - lighted, Gouraud shaded
3 - shaded

II. SIMULATOR DEVELOPMENT AND PERFORMANCE HISTORY

The MPS simulator has evolved from a number of student efforts at the Naval Postgraduate School. In order to understand MPS, we need to discuss its originating systems.

A. FOGM MISSILE SIMULATOR

The Fiber Optically Guided Missile (FOGM) simulator [Ref. 5:p. 10] was originally developed at the Naval Postgraduate School on a Silicon Graphics, Inc. IRIS 3120 graphics workstation. The FOGM simulator allows a user to see the three-dimensional view from the missile as it flies over a fixed 10x10 kilometer area of Fort Hunter-Liggett, California. The missile is able to target, track, and destroy vehicles on the ground. The elevation data for the simulation was provided to the Naval Postgraduate School by the U. S. Army's Combat Developments Experimentation Center (CDEC) at Fort Ord, California. The most difficult task in that project was providing for real-time hidden surface elimination. The IRIS 3120 does not have hardware support for real-time, double buffered hidden surface elimination, so a scanline Painter's algorithm [Ref. 6: p. 39] was used. This algorithm sorts all polygons from farthest away to closest to the viewer's position. All polygons are then drawn in that order to ensure that objects closer to the viewer's eye are not obscured by objects which are further away. There are also algorithms to ensure that vehicles are drawn on top of the terrain, in a proper orientation, and in their proper 100x100 meter grid square. Vehicles had to be drawn after the terrain on which they appeared. When a vehicle was near the boundary between two grid squares, special logic had to make sure the vehicle was drawn after both grid squares. There was no control over the vehicles; they were only given an initial heading and speed.

B. VEH VEHICLE SIMULATOR

The vehicle (VEH) simulator [Ref. 6:p. 8] was also developed on the Silicon Graphics, Inc. IRIS 3120 graphics workstation. It contained the same features as the FOGM simulator for drawing terrain and vehicles. In VEH however, one could actually drive and maneuver a vehicle on the ground in real-time. Only the terrain in the field-of-view was drawn using the Painter's algorithm.

C. FOGM/VEH NETWORKING SIMULATOR

FOGM/VEH NET was a modification of the FOGM and VEH simulators to allow them to network with each other over the Ethernet local area network that links all the graphics workstations together. Networking allowed one user (VEH) to drive a vehicle on one workstation, with the changes in position appearing on the other workstation (FOGM). The missile could also be flown on one workstation and seen on the other workstation.

D. VEH II VEHICLE SIMULATOR

VEH II was the result of moving and modifying the VEH simulator from the IRIS 3120 to an IRIS 4D/70G graphics workstation. Many enhancements were made to the original VEH including the following:

- Complete operation under the MEX [Ref. 7:p. W1] window management system
- Popup menus for all user selected options
- Vehicles could be added to the current "convoy" after driving had commenced
- Pre-saved files of convoys could be entered into the simulator simply by making the correct selection on a popup menu
- The current convoy could be saved to a file for later use, and/or modification
- Multiple processes could be present on the screen simultaneously

VEH II was then modified for the SGI IRIS 4D/70GT hardware and software. Basically the new VEH II had the same capabilities as the old, however some modifications had to be made to allow the program to be compatible with the newer hardware and the 4Sight [Ref. 8] window management system on the 70GT.

E. FOGM, VEH, AND VEH II PERFORMANCE HISTORY

The FOGM, VEH, and VEH II simulators had the capability of representing the field-of-view anywhere from 15 to 55 degrees. In the 15 degree field-of-view, approximately 2000 triangles were drawn. Approximately 8000 triangles were drawn in the 55 degree mode. Since the triangles are different sizes based on their location and the viewer's viewpoint, the number of triangles drawn per second is not an accurate measurement of the performance of the graphics workstation. Instead, the measurement used to compare the performance of the simulators was the number of frames per second drawn. Tables 2.1, 2.2, and 2.3 show the number of frames per second drawn for the three simulators on the three different IRIS graphics workstations.

Since the IRIS 3120 workstation had a relatively slow CPU, only six to eight frames per second of the original VEH simulator could be displayed. After moving the simulator to the IRIS 4D/70G additional enhancements were made, many to improve the user interface. This did not affect the drawing speed, which almost doubled at times as compared to the IRIS 3120. This is essentially due to the faster 10 MIPS processor.

Recall the IRIS 4D/70GT also contains a 10 MIPS CPU, however it contains faster graphics pipeline hardware. The tables show the increase in drawing speed for these workstations. Again, over twice as many frames per second of the VEH II simulation can be displayed. Tables 2.1, 2.2, and 2.3 show the dramatic increase in speed of the vehicle simulator models due to the increase in processor speed and hardware capability.

TABLE 2.1 ONE VEHICLE ON TERRAIN (FRAMES / SECOND)

SIMULATOR/MACHINE	15 DEGREE VIEW	55 DEGREE VIEW
VEH/3120	8.0	6.5
VEH II-4D/70G	14.0	7.0
VEH II-4D/70GT	30.0	16.0

TABLE 2.2 NINE VEHICLES ON TERRAIN IN VIEW (FRAMES / SECOND)

SIMULATOR/MACHINE	15 DEGREE VIEW	55 DEGREE VIEW
VEH/3120	4.0	3.5
VEH II-4D/70G	5.0	3.0
VEH II-4D/70GT	10.0	6.0

TABLE 2.3 NINE VEHICLES ON TERRAIN, NONE IN VIEW (FRAMES / SECOND)

SIMULATOR/MACHINE	15 DEGREE VIEW	55 DEGREE VIEW
VEH/3120	6.0	5.0
VEH II-4D/70G	12.0	7.0
VEH II-4D/70GT	25.0	16.0

III. THE MOVING PLATFORM SIMULATOR (MPS) DESCRIPTION

The Moving Platform Simulator (MPS) is a combination of the FOGM and VEH II simulators with the addition of many new options and advanced capabilities. MPS takes advantage of many features built into the hardware of the Silicon Graphics, Inc. IRIS 4D/70GT. The additional capabilities of the Moving Platform Simulator include:

- Complete operation under the 4Sight [Ref. 8] window management system
- The ability to select any 10 x 10 kilometer grid area from the 35 x 35 kilometer database
- RGB color mode
- User selectable terrain elevation color schemes
- User selectable month and hour to determine the sun's location for light intensity and color
- Realistically lighted vehicles and terrain
- An efficient terrain display algorithm that includes distance attenuation to improve performance while displaying more terrain than earlier models
- Z-buffering for hidden surface removal
- Collision detection
- The ability to track, target, and destroy land vehicles from a FOGM missile
- Broadcast networking to allow multiple simulations to network on different IRIS workstations

A complete discussion of the user interface for MPS can be found in Appendix A. Appendices B and C explain the module and file organization for MPS in detail.

IV. OPERATING AREA

A. FORT HUNTER-LIGGETT DATABASE

The terrain database that the Moving Platform Simulator uses is a small subset of a database provided to the Naval Postgraduate School by the United States Army Combat Developments Experimentation Center (CDEC) at Fort Ord, California. This database consists of elevation and vegetation data in 12.5 meter increments in the area formed by a square with the lower left corner at the Universal Transverse Mercator (UTM) grid coordinate 10SFQ41006000 and upper right corner at 10SFQ77009500. This area is 36 kilometers wide and 35 kilometers high. Each sample consists of 16 bits (two bytes). The three most significant bits are a vegetation code defined in Table 4.1. The remaining 13 bits, after conversion to decimal, is the

TABLE 4.1 VEGETATION CODES FOR DATABASE

VEGETATION CODE		DESCRIPTION
binary	decimal	
000	0	<1 Meter
001	1	1 - 4 Meters
010	2	4 - 8 Meters
011	3	8 - 12 Meters
100	4	12 - 20 Meters
101	5	>20 Meters
110	6	No Data Available
111	7	Not Used at This Time

elevation in **feet** at the point of the sample. The complete database consists of 6400 (80^2) samples per square kilometer, 1260 (35*36) square kilometers, and two bytes per sample which results in a file that contains 16,128,000 (6400*1260*2) bytes.

The Moving Platform Simulator is designed to handle a 35x35 kilometer area with a resolution of 100 meters. Therefore, the database from CDEC was reduced down from its original size to a file of 245,000 bytes. There are 100 (10^2) samples per square kilometer, 1225 (35*35) square kilometers and two bytes per sample which yield 245,000 (100*1225*2) bytes. The vegetation information was ignored since most of the codes indicated that the information was unknown.

B. SELECTION METHODOLOGY

The Moving Platform Simulator is designed to allow a user to select any 10x10 kilometer area from the 35x35 kilometer database. The 10x10 kilometer restriction is due to the restriction in the original version of the Vehicle Simulator (VEH) [Ref. 6:p. 15]. A logical modification to MPS is to eliminate this restriction.

Also, the user is restricted to selecting an area that begins and ends on a one kilometer grid line. As an example, he cannot select an area beginning at 10SFQ44246868. The user is restricted to selecting an area where the two sets of four digit numbers after the 10SFQ are multiples of 100. Therefore, the user is able to select one of the following: 10SFQ44006800, 10SFQ45006800, 10SFQ44006900, or 10SFQ45006900.

C. GLOBAL COLOR SCHEME

The base color (before lighting) of the terrain polygons is selected by the user from a popup menu of available choices. The default coloring scheme is a brown ramp with eight primary colors. The colors in the brown ramp range from almost black to a deep brown. Black indicates the highest elevations and the brown the lowest. The color to use for any particular elevation is selected from these eight colors by using a linear function shown in Figure 4.1. Notice that the manifest constants MIN_ELEV and MAX_ELEV are defined in the header file mps.h and represent the absolute minimum

```
/* Creates a linear 'ramp' of elevations between the min and max */
 for(i=0;i<=7;i++)
        bkpt[i] = (MINELEV + i + 1) * ((MAXELEV - MINELEV) / 8);
```

FUNCTION AS IT APPEARS IN MPS

MINELEV = zero meters
MAXELEV = 1134 meters

MANIFEST CONSTANT VALUES

```
for(i=0;i<=7;i++)
        bkpt[i] = (i + 1) * (1134 / 8);
```

FUNCTION WITH MANIFEST CONSTANT VALUES SUBSTITUTED

bkpt[0] = 1 * 1134 / 8 = 141
bkpt[1] = 2 * 1134 / 8 = 283
bkpt[2] = 3 * 1134 / 8 = 425
bkpt[3] = 4 * 1134 / 8 = 567
bkpt[4] = 5 * 1134 / 8 = 708
bkpt[5] = 6 * 1134 / 8 = 850
bkpt[6] = 7 * 1134 / 8 = 992
bkpt[7] = 8 * 1134 / 8 = 1134

RESULTING VALUES FOR bkpt[]

Figure 4.1 Elevation Breakpoint Function

and maximum elevations in the entire CDEC terrain database.[1] The function divides the difference into eight equal distances and then repeatedly calculates the proper value eight times which yields a linear ramp of breakpoints between the different colors in the color ramp. The breakpoints are stored in a short array. The functions *display_big_map*() and *display_map*() use this array of breakpoints to display the two dimensional maps, *drawterrain*() uses it to draw the three dimensional terrain, and *display_legend_for_big_map*() and *display_legend_for_navbox*() use it to draw the legends for the two dimensional maps.

In addition to the eight primary colors discussed above, each color ramp is augmented with eight secondary colors for the purpose of checkerboarding the three dimensional terrain. Each primary color has a corresponding secondary color that is slightly darker. When more than one terrain square falls into the same elevation breakpoint, then every other square is drawn in the secondary color. This gives the terrain relief and depth when large areas are at similar elevations.

---

[1] This function can be found in the file display_big_map.c.

V. GRAPHICS DISPLAY SPECIFICS

A. PICTURE COMPLEXITY

After completing the introductory and main menus, the main event loop of the simulation begins. This portion is further divided into driving and flying sections.

While driving a platform, other platforms and the terrain are displayed on the screen. The view of the world is from the driver of the selected platform. Since it is computationally expensive to display graphical objects which do not appear in the field of view, only the objects and terrain that can be seen are displayed. A platform such as a covered jeep is composed of approximately 50 polygons .

Four graphical windows are visible while driving a vehicle. These are illustrated in Figure 5.1. The platforms and terrain are present in the map window, which is the same window used to display the initial terrain maps. The menu window, which is located in the upper right hand area, must be updated each display frame and contains performance and simulation information such as:

- Current frames per second drawing speed
- Average of the last 100 frames per second values
- Current hour of the day for light location and intensity
- Current month of the year for light location and intensity
- Current sunrise, midday, and sunset times
- Total number of polygons that are being displayed in the map window

The navigation window appears in the middle right hand side and contains a smaller picture of the 10 x 10 kilometer operating area. Since this area is fixed during platform operations, it is only drawn once in the color scheme selected by the user. Using the overlay drawing capability, a blue arrow and corresponding V shaped lines alert the user to the course and field of view of the currently controlled platform. These lines can be erased and redrawn each frame without having to redraw the entire map.

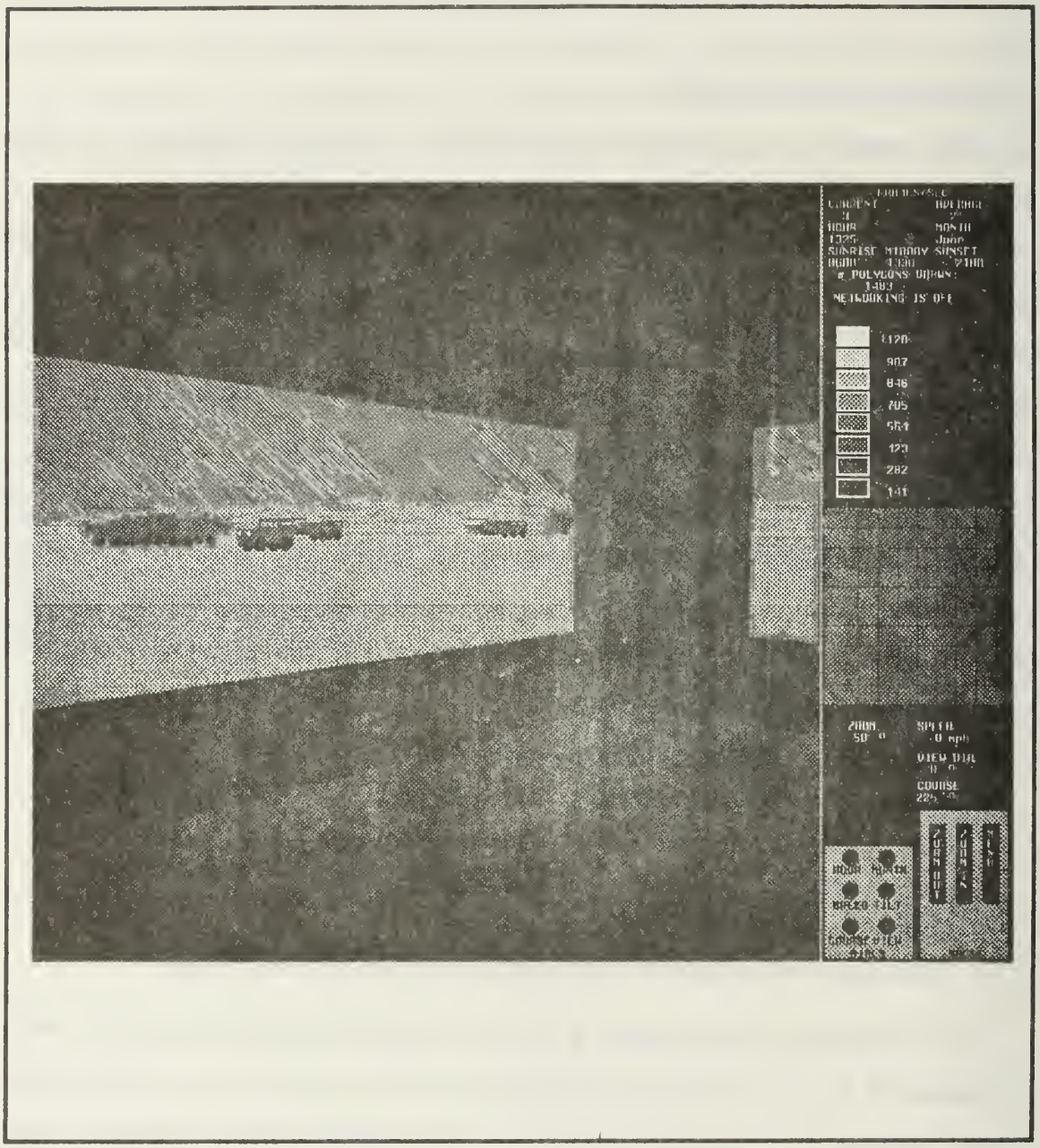

Figure 5.1 View from a Vehicle

Above the map display is a legend to equate the terrain color to the corresponding elevation value.

The indicator window appears in the bottom right hand side and contains information about the platform that is currently being controlled. This information includes:

- Velocity
- Course direction
- View direction
- Zoom angle for field of view
- Dial help information

All of these quantities must be updated for each frame. Although there is no measurement of the tilt angle, the user can also adjust the tilt of the eye with the appropriate dial.

With exception of added detail, the complexity of the picture while flying a missile is similar to that of driving a vehicle. This is shown in Figure 5.2. Other vehicles and missiles can be visible with the terrain in the map window. This simulates the view an operator has from the camera on the missile. In addition, the tilt and pan angle values are displayed along a slider bar in the map window. These are controlled using the mouse. In the center of the map window is a rectangular box formed by four white dots. This defines the tracking area in which a vehicle must be seen in order to be targeted. While a vehicle is being tracked by the missile, the dots flash white and red and the tracked vehicle type is displayed toward the top of the map window.

The menu and navigation windows contain the same information as for the driven vehicles. The indicator window, however, displays different quantities. If a vehicle is not being tracked, the following quantities are updated and displayed for each frame:

- Velocity
- Course direction
- Zoom angle for field of view
- Altitude to ground level

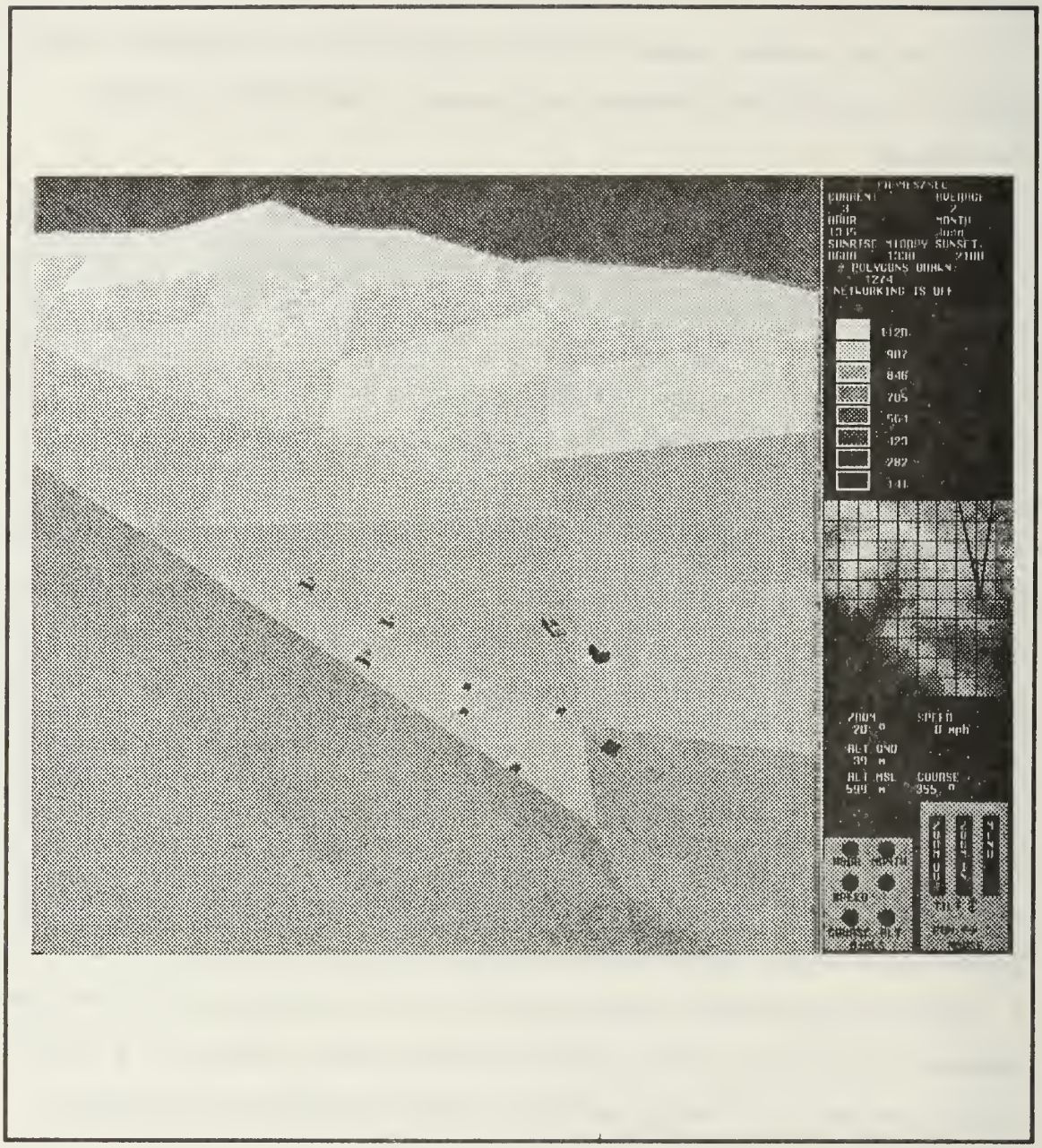

Figure 5.2 View from a Missile

- Altitude of the missile to the terrain
- Dial help information

When the missile is locked onto a vehicle, the user cannot activate all the functions described above. This is also reflected in the information given to him in the indicator window. Figure 5.3 shows the view from a missile as it is locked onto a covered jeep. The user can no longer change the missile's course, pan view, tilt view, or altitude. Additional information is present in the window to keep the user informed of the current distance between the missile and the target.

B. INCORPORATED GRAPHICS TECHNIQUES

Many graphics techniques are incorporated in the simulation to produce the images which are displayed in each window. These techniques take advantage of the hardware to achieve the necessary speed while performing the desired task.

1. Double buffering

Double buffering is a technique where the picture to be displayed is drawn in an area of memory that is not currently being displayed (the back buffer). After the picture is completed, the function *swapbuffers*() displays the completed picture. This gives the appearance of smooth motion since the user sees only the changes from frame to frame.

The buffer must be cleared before each picture is drawn. If single buffering were used, the screen would flash repeatedly and the appearance of smooth motion would be impossible.

2. Z-buffering

Z-buffering [Ref. 9:pp. 262-264] is one way of achieving hidden surface elimination. Recall that the earlier simulations relied on a scanline Painter's algorithm to order the objects in the scene from farthest away to closest to the viewer's eye. The scanline Painter's algorithm was complicated to implement and consumed the

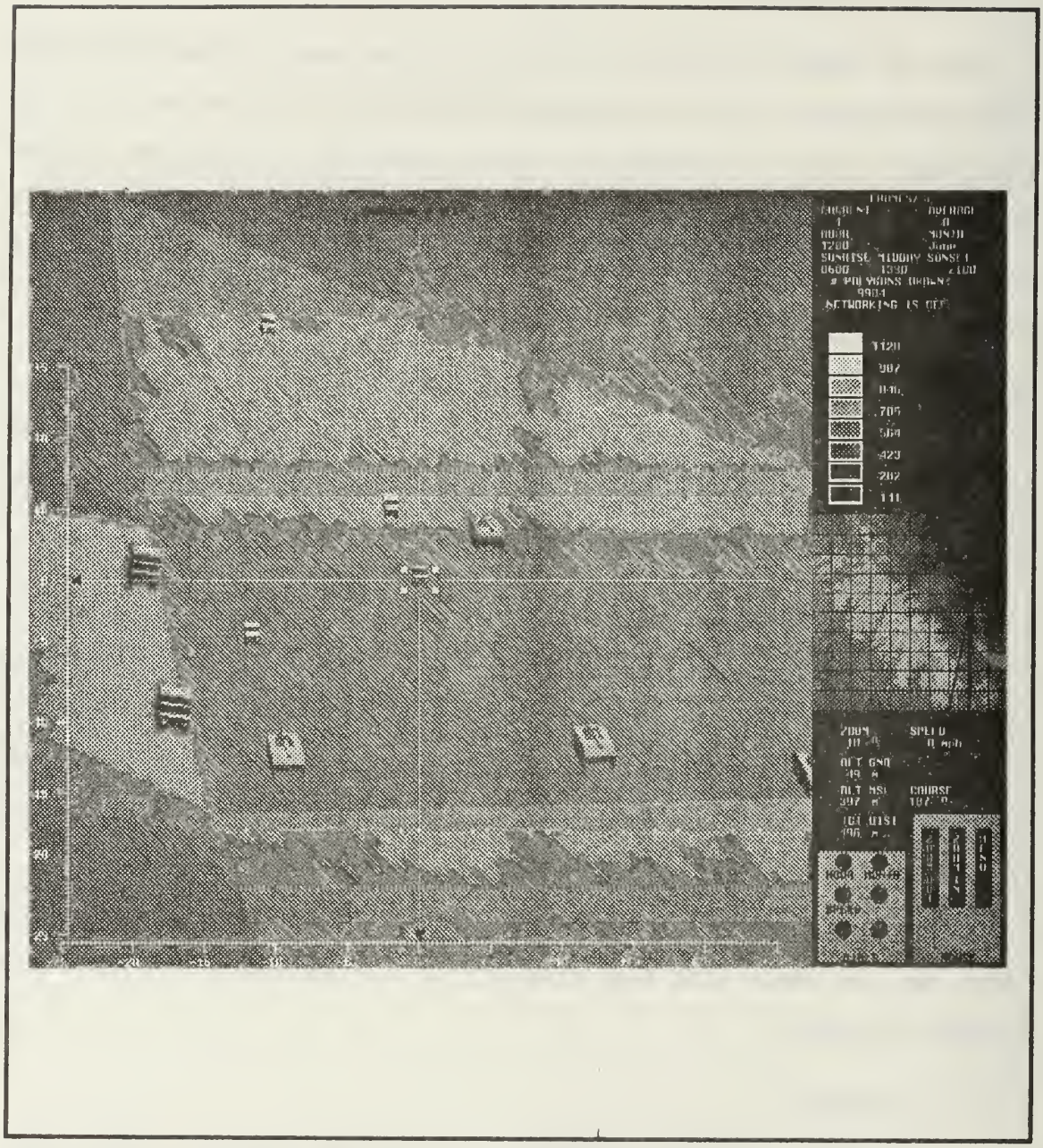

Figure 5.3 Missile Locked Onto a Covered Jeep

available CPU capacity. This technique was utilized because the earlier workstations did not have the capability of supporting Z-buffering and double buffering simultaneously.

The concept of Z-buffering is quite simple, and the hardware implementation allows it to execute quickly. The coordinate system is constructed so that the z-axis is perpendicular to the plane of the screen. Initially, the z-depth is set with an *lsetdepth*() command from zero to hexadecimal 7fffff, the largest value possible for the 24 bits of Z-buffer memory available. Each time the terrain and platforms are to be drawn, Z-buffering is activated using the *zbuffer*(TRUE) command, and the Z-buffer is cleared using the *zclear*() command. The *zclear*() function sets the Z-buffer to the farthest away value (7fffff).

When all of the polygons comprising the desired terrain and platforms are drawn, the system keeps track of the z-value for each filled pixel on the screen. Only the pixel with the closest z-value is displayed. On the IRIS 4D/70GT, these is no time penalty for using Z-buffering, and its use greatly simplifies the display algorithm.

3. RGB Color

Earlier simulations used color map mode by defining red, green, and blue color values for each index into a color lookup table. The desired color was selected by setting the color to the appropriate index. This method does not work with the real-time lighting and shading techniques of the IRIS 4D/70GT, so the simulation was converted to RGB color mode.

Instead of defining indices into a table, the desired color is activated with a *RGBcolor*(r,g,b) call after the function *RGBmode*() is called once during the initialization phase of the model. The parameters r, g, and b are short integer values from zero to 255 and define the amount of red, green, and blue components desired. *RGBcolor*(255,255,255) is white, while *RGBcolor*(0,0,0) is black. Besides RGB color mode,

the IRIS 4D/70GT's lighting capabilities are used in the simulation. Details of the lighting capabilities used are discussed in a later section.

4. Perspective World Views

A world coordinate system is defined in each window of the simulation. For two dimensional views, this involves using the *ortho2*() command with its appropriate parameters. The values of the parameters are chosen for convenience, since all drawing is performed in the world coordinate system.

In order to give the appearance of a three dimensional view in the map window while operating a platform, *perspective*() and *lookat*() commands are used. The *perspective*() command is called with the following arguments:

- Field of view angle
- Ratio of x to y of window
- Near clipping plane value
- Far clipping plane value (look distance)

The field of view is selected by the user. The ratio of the x to y of the map window is one since the window is chosen to be a square. The near clipping plane value is 0.1, because zero does not work with Z-buffering. The far clipping plane is chosen to be large.

The *lookat*() command is called with the following arguments

- X of viewer's eye
- Y of viewer's eye
- Z of viewer's eye
- X of position to focus upon (px)
- Y of position to focus upon (py)
- Z of position to focus upon (pz)
- Twist angle rotation about z-axis

The viewer's x, y, and z position is the location of the platform being operated. The y-coordinate is adjusted for the height of the driver's eye above the base of the

platform. The position to focus upon is a function of the platform's location, look distance, and tilt angle. Figure 5.4 shows the computations for these quantities. The twist angle is set to zero.

```
/* compute coordinate of where camera is looking */
sine = sincos(lookang,&cosine);
*px = driven->x + cosine * MAXLOOKDIST;
*pz = driven->z - sine   * MAXLOOKDIST;

if(tilt > 0.0)
        *py = (TILT_FACTOR * tilt) * driven->y;
else
        *py = 0.0;
```

Figure 5.4 Computations of px, py, and pz

C. INCORPORATED MODEL TECHNIQUES

1. Platform Attitude Update

Before each frame is displayed in the map window, all the platforms must be updated to account for their movement from the previous frame. This involves updating its position on or above the terrain, its grid position, and the viewer's look position.

MPS uses the same algorithm as the one in VEH [Ref. 6:p. 72] to update the position and grid position of each platform. Briefly, this involves computing the new position as a function of the platform's velocity and elapsed time since the last update. The platform's tilt angle (rotation around x-axis) and incline angle (rotation around z-axis) are also computed based on the orientation of the terrain beneath it. The platform's heading is a function of its course, which is saved in the structure for

each platform. A complete description of the computations of these angles can be found in [Ref. 6:pp. 70-78].

A linked list is maintained for each grid square to keep track of the platforms currently in each square. As the platforms move, their x and z positions change. Since there are 100 grid squares in each direction (x and z), the updated x and z positions of each platform are divided by 100 and truncated to determine the new grid square in which the platform resides. Then the platform is removed from the previous grid square list and placed in the list pointed to by the new grid square. These lists simplify the displaying of the platforms. After drawing a given grid square, its pointer is followed to the list of platforms to be displayed within the grid square.

The function *update_look_pos*() and *update_look_pos_fogm()* are used to update the viewer's look position. This operation involves setting the position upon which to focus in the *lookat*() command.

2. Target Selection and Lock

Target selection is only allowed when a user is operating a FOGM missile and there is at least one ground platform active in the simulation. A ground platform is defined to be one of the following types of platforms:

- Tank
- Truck
- Jeep - Covered
- Jeep - Open

The ground platform targeted may either be one specified interactively or obtained from another process if networking is currently active.

The actual process of selecting a target is performed by moving the mouse until the desired target is in the crosshairs of the missile. Figure 5.3 shows a picture of a covered jeep inside the crosshairs of a FOGM missile. Zooming the camera in and out assists the user in placing the crosshairs on the target.

When the target is inside the crosshairs, and the user desires to lock onto it, he must push the right mouse button and select TOGGLE TARGET TRACKING. The functions in the file *tracking_check.c*() determine if a platform was in fact inside the crosshairs of the FOGM missile. If one is found, MPS performs the actions necessary to lock onto the target.

The actual algorithm that determines if a platform is in the crosshairs of the missile involves the IRIS pick capability. That capability allows a programmer to easily determine if a picture element is within a desired rectangular window. If one or more picture elements are present, unique identification numbers are returned in an array as well as the number of elements that were detected. By defining the rectangular window to be where the crosshairs are, the function is able to accurately determine if a platform is inside the crosshairs.

3. Missile Tracking Update

After the user has targeted and locked onto a platform, the missile must track and destroy it. To do this, MPS takes control of the altitude, course, and nose camera movement controls. The user still has control of all other input devices. The nose camera tilt and pan angles are set to zero when a platform is targeted and locked onto so only the altitude and course need to be adjusted automatically by MPS. The function that performs the necessary calculations to update these parameters is *handle_tracking*(). The algorithm that is used has basically two parts. The first part takes the distance that the missile traveled in this frame update and calculates the necessary change in the altitude of the missile. The second part of the algorithm changes the course so that it coincides exactly with the location where the targeted platform will be after the frame update. The code for this algorithm is in Figure 5.5. In addition to this algorithm, the *lookat*() function is set so that the user's point-of-view is exactly pointed at the center of mass of the targeted platform. This is done in the function *update_look_pos_fogm*().

```
/* Find the distance that the missile traveled in relation to the
   platform that it is tracking.                                    */
deltax = temp->x - temp->track->x;
deltaz = temp->z - temp->track->z;

/* Find the angle that the missile needs to point to track the platform  */
angle = (float)(atan2((double)(deltaz),(double)(-deltax)));
if (angle < 0.0) angle += TWOPI;

/* Set the angle and course for the missile  */
temp->ang = angle;
temp->cse = ((angle*RTOD) <= 90.0) ? (90.0 - (angle*RTOD))
                  : (450.0 - (angle*RTOD));

/* Calculate the distance the missile traveled in this frame  */
distance_missile_traveled = temp->vel*elapsedsec;

/* Calculate the ground distance to the tracked platform  */
ground_distance_to_target =
                    ((float)(hypot(((double)(deltax)),((double)(deltaz)))));

dist_ratio = distance_missile_traveled/ground_distance_to_target;
ground_level = gnd_level(temp->track->x,-temp->track->z);

/* Calculate the new altitude for the FOGM missile  */
/* The 0.35 factor is a number that made the function look most realistic  */
temp->alt -= 0.35 * (temp->alt - ground_level) * dist_ratio;
```

TWOPI = Two Pi = 6.2831853
RTOD = Radians To Degrees = 57.295779

Figure 5.5 Tracking Update Algorithm

4. Indicator Displays

The Moving Platform Simulator involves many functions and operations that require information to be displayed to the user. Most of this information is displayed in a window in the upper right portion of the simulation. An example of this information is depicted in Figure 5.6.

The first line of data indicates the frames per second (FPS) in two different ways. The left number is the FPS for the current frame. The right number is the FPS for the last 100 frames.

The next line indicates the hour of the day and the month of the year. These determine where the sun is located, what color it is, and its intensity.

The next line indicates the time of day that sunrise, sunset and midday occur for the current month. This information is helpful when setting the time of day to view the lighting effects in the morning, evening, and midday.

The next line indicates the number of polygons that were drawn to generate the three dimensional picture. This number includes not only the polygons that make up the terrain but also the polygons in all the platforms.

Other information that is displayed during the simulation include the following:

- If you are tracking a platform, the type of platform you are tracking is displayed in the upper center of the viewing area.
- If you are a tracked platform, a message indicating this is displayed in the upper center of the viewing area.
- When operating a missile, the tilt and pan angles of the nose camera are indicated on slider bars on the left and bottom of the viewing area.
- Superimposed on top of the small two dimensional map on the right side of the screen is a small blue arrow indicating the course of the driven platform, and a larger blue **V** indicating the current field-of-view.
- Also superimposed on the two dimensional map are blue dots indicating all other ground platforms currently in the simulation and black dots indicating FOGM missiles in the simulation.
- In the lower right of the simulation is a depiction of the mouse and dials with labels indicating the function of each. Also, there are indicators such as course, speed, altitude, etc. Figure 5.7 shows the indicators for a ground platform.

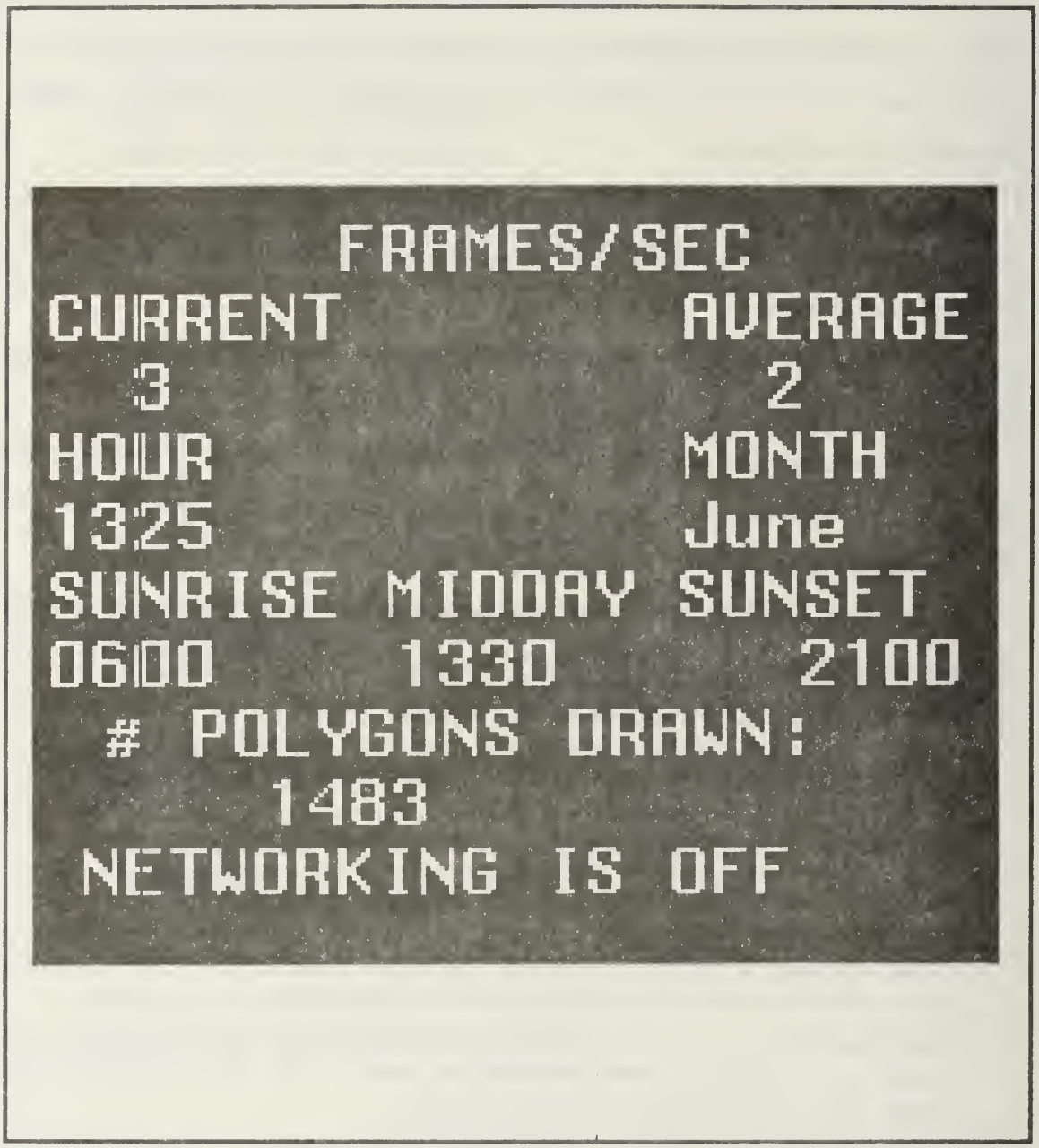

Figure 5.6 Example of the Data Displayed During the Simulation

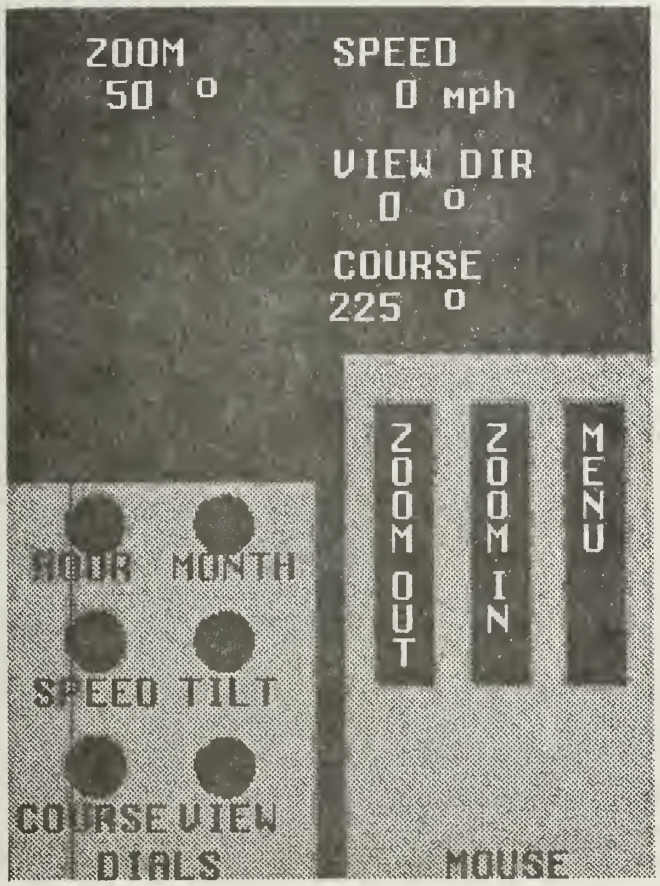

Figure 5.7 Indicators For a Ground Platform

Figure 5.8 shows the indicators for a FOGM missile that is NOT tracking a platform and Figure 5.9 shows the indicators for a FOGM missile that IS tracking a platform.

5. Collision Detection

Collision detection was implemented to increase the realism of the Moving Platform Simulator. When any two platforms, including obstacles/wrecks, are closer than an arbitrary distance (currently set to 5 meters) then both platforms are destroyed and changed to wrecks. If one of the platforms was the driven platform then the user is returned to the main menu and the platform he was operating as well as the one he hit are turned into wrecks.

The function that implements collision detection is *collision_detection*(). Figure 5.10 outlines the algorithm for collision detection.

D. LIGHTING AND GOURAUD SHADING

1. Light Intensity, Location, Color

The Moving Platform Simulator contains real-time realistically lighted platforms and terrain using hardware routines for lighting and Gouraud techniques for shading. In MPS, the user can select any hour, minute, and month under which to operate. A sunrise time and total number of daylight hours are defined for the months January, June, and December. From these times the sunrise, sunset, and midday times are computed for the current month selected by linear interpolation.

If the user selects a time that is later than sunset or earlier than sunrise then the attributes defining the light are set to approximate night conditions. Since the sky is drawn independent of the lighting model its color is also changed for night by calling *RGBcolor*() with pre-defined night colors.

The color of the sky is also changed during the daylight hours. To simulate the morning and evening skies the sky's color is set to a reddish tint for one hour after sunrise or one hour before sunset. At other times during the day, the sky's red, green,

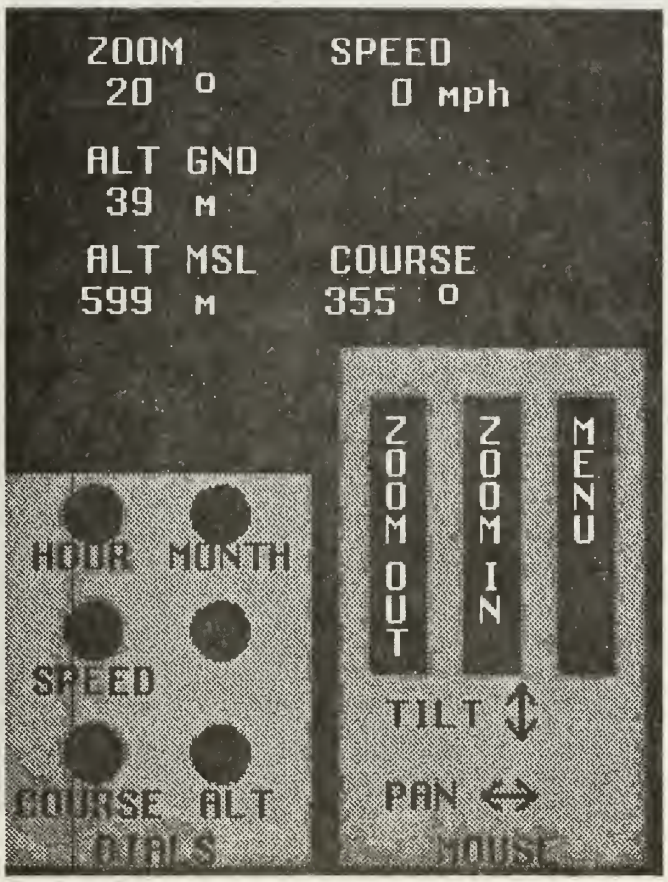

Figure 5.8 Indicators For a FOGM Missile That is Not Tracking

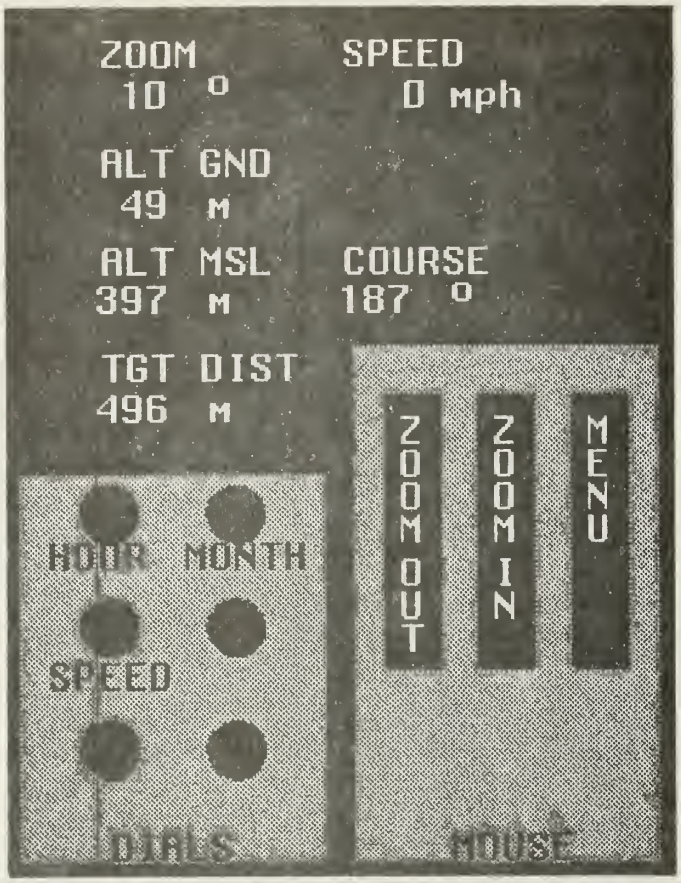

Figure 5.9 Indicators For a FOGM Missile That is Tracking

```
while (currveh != NULL)
{
    if (currveh->t != WRECK)
    {
        /* Check for platform OUT OF BOUNDS. */
        if ( currveh->x <= FUDGE || currveh->x >= 10000.0-FUDGE ||
           -currveh->z <= FUDGE || -currveh->z >= 10000.0-FUDGE)
        {
            numveh[currveh->t]--;
            numveh[WRECK]++;
            currveh->t = WRECK;
            currveh->vel = 0.0;
            if (currveh == driven)
            {
                *event_status = EVENT_START;
                explosion();
            }
        }
    }

    /* Check for collisions between platforms */
    checkedveh = currveh->next;
    while(checkedveh != NULL)
    {
        if(there_is_a_collision(currveh,checkedveh))
            kill_the_platforms(NON_NET,currveh,checkedveh,event_status);
        checkedveh = checkedveh->next;
    }
    currveh = currveh->next;
}

#define FUDGE        150.0
#define MIN_SEPARATION   5
```

Figure 5.10 Collision Detection Algorithm

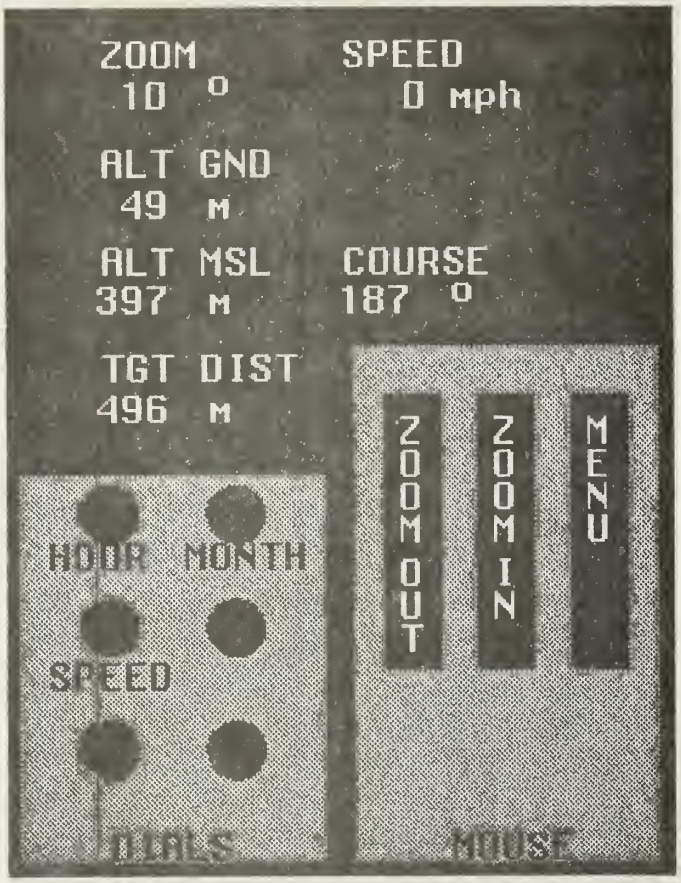

Figure 5.9 Indicators For a FOGM Missile That is Tracking

```
while (currveh != NULL)
{
   if (currveh->t != WRECK)
   {
      /* Check for platform OUT OF BOUNDS. */
      if ( currveh->x <= FUDGE || currveh->x >= 10000.0-FUDGE ||
         -currveh->z <= FUDGE || -currveh->z >= 10000.0-FUDGE)
      {
         numveh[currveh->t]--;
         numveh[WRECK]++;
         currveh->t = WRECK;
         currveh->vel = 0.0;
         if (currveh == driven)
         {
            *event_status = EVENT_START;
            explosion();
         }
      }
   }

   /* Check for collisions between platforms */
   checkedveh = currveh->next;
   while(checkedveh != NULL)
   {
      if(there_is_a_collision(currveh,checkedveh))
         kill_the_platforms(NON_NET,currveh,checkedveh,event_status);
      checkedveh = checkedveh->next;
   }
   currveh = currveh->next;
}

#define FUDGE        150.0
#define MIN_SEPARATION  5
```

Figure 5.10 Collision Detection Algorithm

and blue (RGB) components are computed as a function of the current time. RGB components of the sky are defined for sunrise, midday, and sunset. The actual sky color is found by linear interpolation between these values.

On the IRIS 4D/70GT, light sources are described by defining their ambient color and position. RGB values are defined for the light source in MPS at sunrise, midday, and sunset. Then the RGB components of the light's ambient characteristics and color are found by linear interpolation using the current time. To account for the difference in months, a factor is also computed for each month. This is used to multiply each component and to have the light (sun) appear with less intensity in the winter months and more intensity in the summer months.

The determination of the sun's location and color gives MPS a realistic lighting model. The terrain and vehicles appear brighter at noon and darker toward sunrise and sunset. The time between sunrise and sunset is longer in June and shorter in December. Be aware the actual values chosen for sunrise and number of hours in the day for each month are not based on any real data. Instead they are chosen so that longer days appear in the month of June and shorter days appear in the month of December. During other months the sunrise time and number of hours in the day are found by linear interpolation.

The sun appears to move from the positive x-position (east) to the negative x-position (west) as time changes from sunrise to sunset. The first quantity computed is the sun's z-position in the sky at noon. This is a function of the month, with the sun at 5000.0 in the z direction in January, zero in June, and 6000.0 in December. At noon both the y-position and x-position are zero. This means at noon the sun is directly over the origin in June and farthest away from the origin in December.

The radius of the sun about the x-axis is computed next as a function of month. The pre-defined values for the sun's radius are 7500.0 in January, 10000.0 in June, and 7000.0 in December. These values give the appearance of longer days in

June, since the sun travels from 10000.0 to -10000.0 and shorter days in December, since the sun travels from 7000.0 to -7000.0.

Figure 5.11 shows the ground track the sun makes as it moves across the sky in the months of June and December. The paths during the other months fall in between these paths. The x-position of the sun is computed using the current time of day and radius. Then the y-position is defined by solving the equation of a circle. For example, if the current time is sunrise in June, the x-position would be 10000.0, the y-position would be zero, and the z-position of the sun would be zero. The sun's position is normalized for the graphics' lighting model.

After defining all parts of the sun in the array **light**, it is defined with a *lmdef*() command and bound with a *lmbind*() command as follows:

- *setwindow*(MAPWIN);
- *lmdef*(DEFLIGHT,MYLIGHT,14,light);
- *lmbind*(LIGHT0,MYLIGHT);

Note that this must be done in the window in which the light is to be used. The *lmdef*() command equates the light's characteristics with a manifest constant MYLIGHT. The *lmbind*() binds the light's characteristics to light number zero.

2. Platform Lighting

Material characteristics must be defined for each polygon in order to use the lighting model. This includes defining the emitted, diffuse, and ambient parts for the RGB components. In MPS an infinite light source is used since a local light is not computationally free.

A very small amount of emitted color is defined for each platform in order to make night viewing easier. However the emitted red of the rear lights is set to one, and the emitted RGB components of the headlights are set to one (white).

The diffuse portion of the material describes how much light is reflected from the direct sun. Recall there are approximately 50 polygons grouped to form each

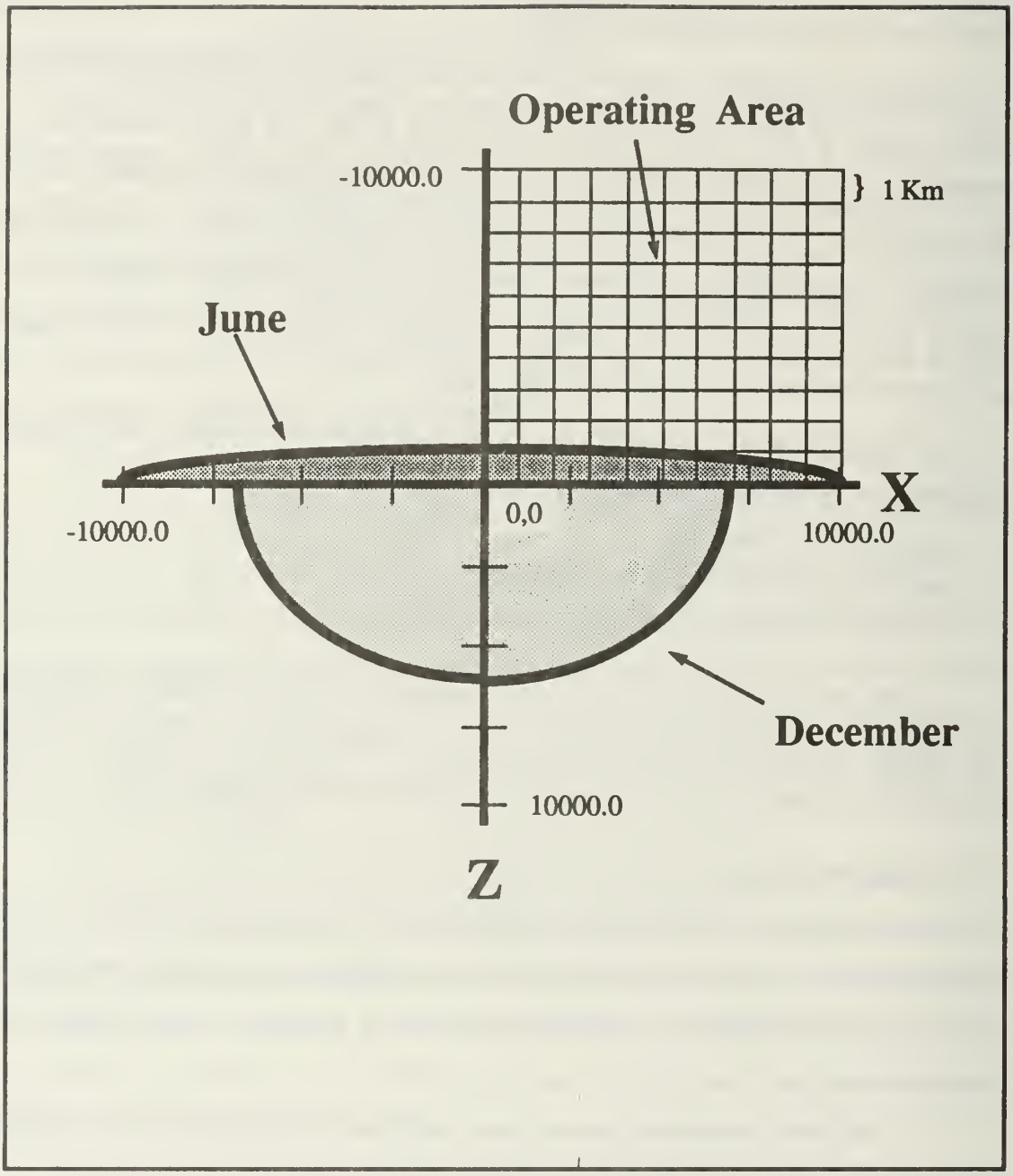

Figure 5.11 Ground Track of Sun

platform. All the outside polygons of each platform are defined to have the same diffuse qualities. Likewise, all the inside polygons are defined with identical diffuse qualities, but these qualities are not the same as those used for the outside polygons.

The ambient portion of the materials describes how much light is reflected from the material that is not in direct sunlight. Again all the outside polygons are defined to have one characteristic, while all the inside polygons have another. No work was done to explore the possibility of changing the diffuse or ambient qualities of specific polygons of the platforms. This could give a more realistic appearance to the platform and is discussed in the chapter concerning future capabilities (Chapter VIII).

Normal vectors are computed for each polygon of the platforms during initialization. During operation, the amount of light reflected from a platform's polygon is a function of the angle between the polygon's normal vector and the light source using Lambert's Cosine Law [Ref. 9:p. 278].

3. Terrain Lighting

Real-time lighting models were not used in earlier simulations so the terrain was drawn using a checkerboard effect, varying the shades of green as a function of a fixed light source. The terrain's color in MPS is drawn to have the same color as the large map display and the terrain's intensity changes as the sun moves across the sky.

For each possible color scheme for the terrain, a scale of eight major colors is used to ramp from low to high elevation. To display the terrain in a checkerboard pattern eight more minor colors are defined, each being slightly darker than the original eight. When the ten by ten kilometer grid and current color scheme are selected by the user, each even grid square is labeled with a major color, and each odd grid number is given a minor color.

Each grid square must also be given emitted, ambient, and diffuse characteristics for the RGB components. All emission quantities are zero, ambient quantities for

the red, green, and blue components are 0.5, and diffuse quantities are set to the color defined by the elevation scaled from zero to one.

If the selected ten by ten kilometer area contains water, a special water material is used. Also a ground plane is drawn with the material characteristics of the center grid square (50,50) of the operating area. This area surrounds the operating area to avoid the appearance that the world ends at the edge of the map.

4. Lighting Summary

The real-time lighting model available on the IRIS 4D/70GT produces realistically lighted platforms and terrain. It is also confusing at times to follow its implementation. Figure 5.12 gives a simplified view of the steps necessary to activate the lighting model. The purpose of Figure 5.12 is not to define all the functions used but to give an overall view of the steps needed to activate the lighting model. The reader is directed to the actual MPS program source code and the IRIS User's Guide [Ref. 10] for more details.

E. TERRAIN DISPLAY

MPS contains a completely new algorithm for displaying the terrain. All the terrain within the field of view from the driven position to the edge of the map is displayed. A distance attenuation procedure is used to speed up the time needed to display the terrain. Under menu option, it is also possible to draw all the grid squares in a detailed 100 x 100 meter mode. This section describes the data structure and display algorithm.

1. Data Structure

After the user selects the ten by ten kilometer operating area, all the data points within the area are extracted from the database. This involves defining all vertices of the two triangles comprising each 100 x 100 meter grid square. The algorithm is defined in detail in [Ref. 6:p. 15].

```
/* initialize light model */
    define materials for polygons
    define sun location for light definition
    bind light
    compute polygon normals

/* draw platforms and terrain */
    turn on Z-buffering option
    clear Z-buffer
    set mmode to MVIEWING
    bind light model
    load unit vector on system stack
    call perspective and lookat
    draw terrain
    draw platforms
    set mmode to MSINGLE
    unbind light model
    turn off Z-buffering option
```

Figure 5.12 Lighting Model Summary

2. Display Algorithm

Before the terrain is displayed, the area within the field of view is determined (see Figure 5.13). To ensure that enough of the terrain is drawn from the driven position to the edge of the map, the driven position's x and z coordinates are offset by values which are a function of the field of view. A larger offset is needed for smaller fields of view.

Using the offset position and field of view angle, parallel lines are constructed. The intersections of the new left and right view lines with the edge of the map are computed. All grid squares within the bounded area are displayed as shown in Figure 5.13.

Let (lx,lz) be the grid square of the left field of view line and (rx,rz) be the grid square of the right field of view line. Grid square (x,z) is the grid square corresponding to the offset driven position. Note that all z grid squares are chosen to be positive values.

The algorithm begins by computing the initial starting and stopping x and z values based on the following criteria:

- xstart is the minimum of lx, x, rx
- xstop is the maximum of lx, x, rx
- zstart is the minimum of lz, z, rz
- zstop is the maximum of lz, z, rz

A test is performed if the field of view is greater than 90 degrees. This can cause invalid limits to be computed as shown in Figure 5.14. The test shown in Figure 5.15 is used to make sure the terrain is displayed to the end of the map.

After the grid squares for the left and right view angles are defined, the look direction is computed based on the current look angle. If the look angle is greater than 45 degrees and less than 135 degrees, the direction is north. If it is greater than or equal to 135 degrees and less than or equal to 225 degrees, the direction is west. If

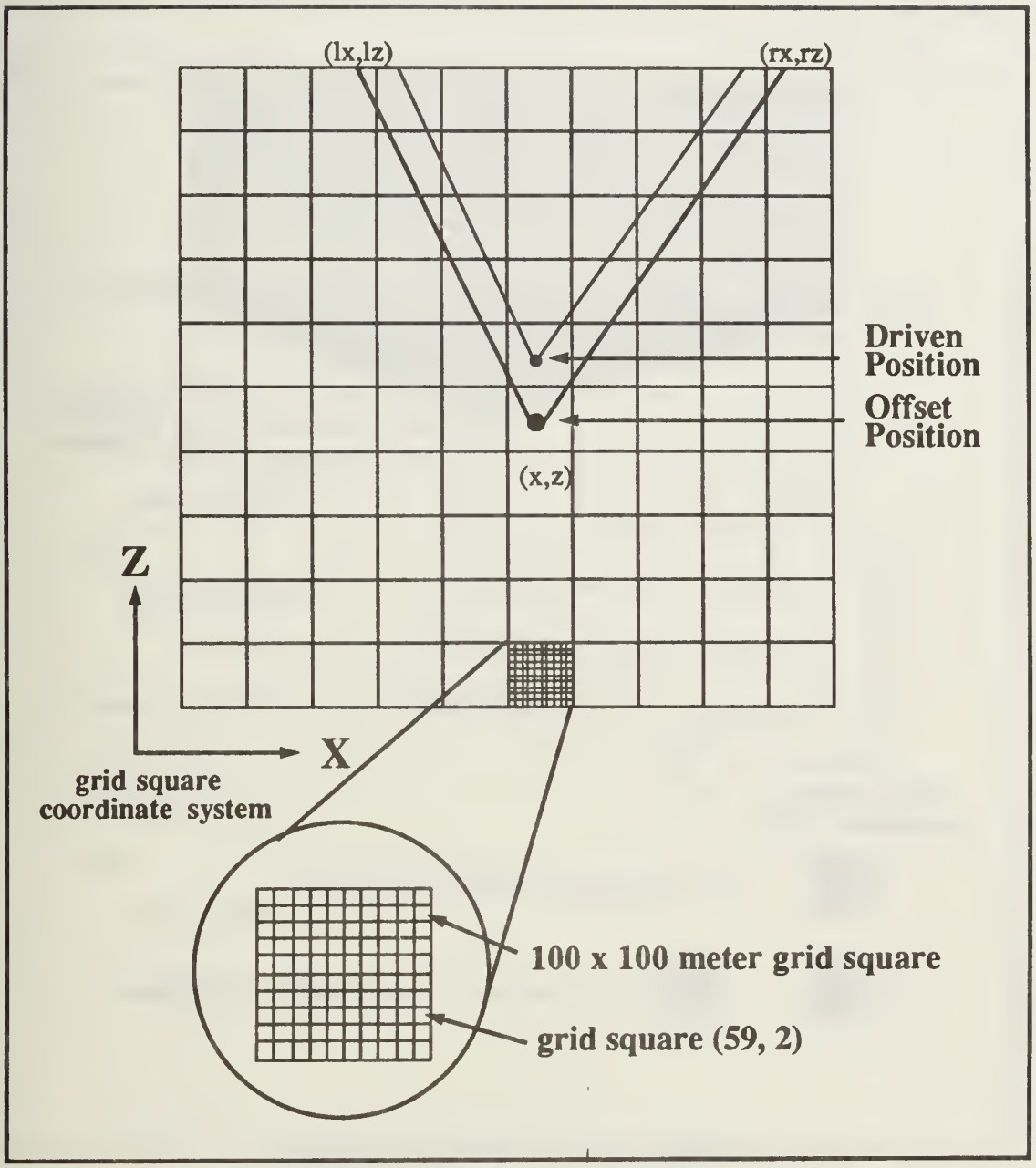

Figure 5.13 Field-of-View Display

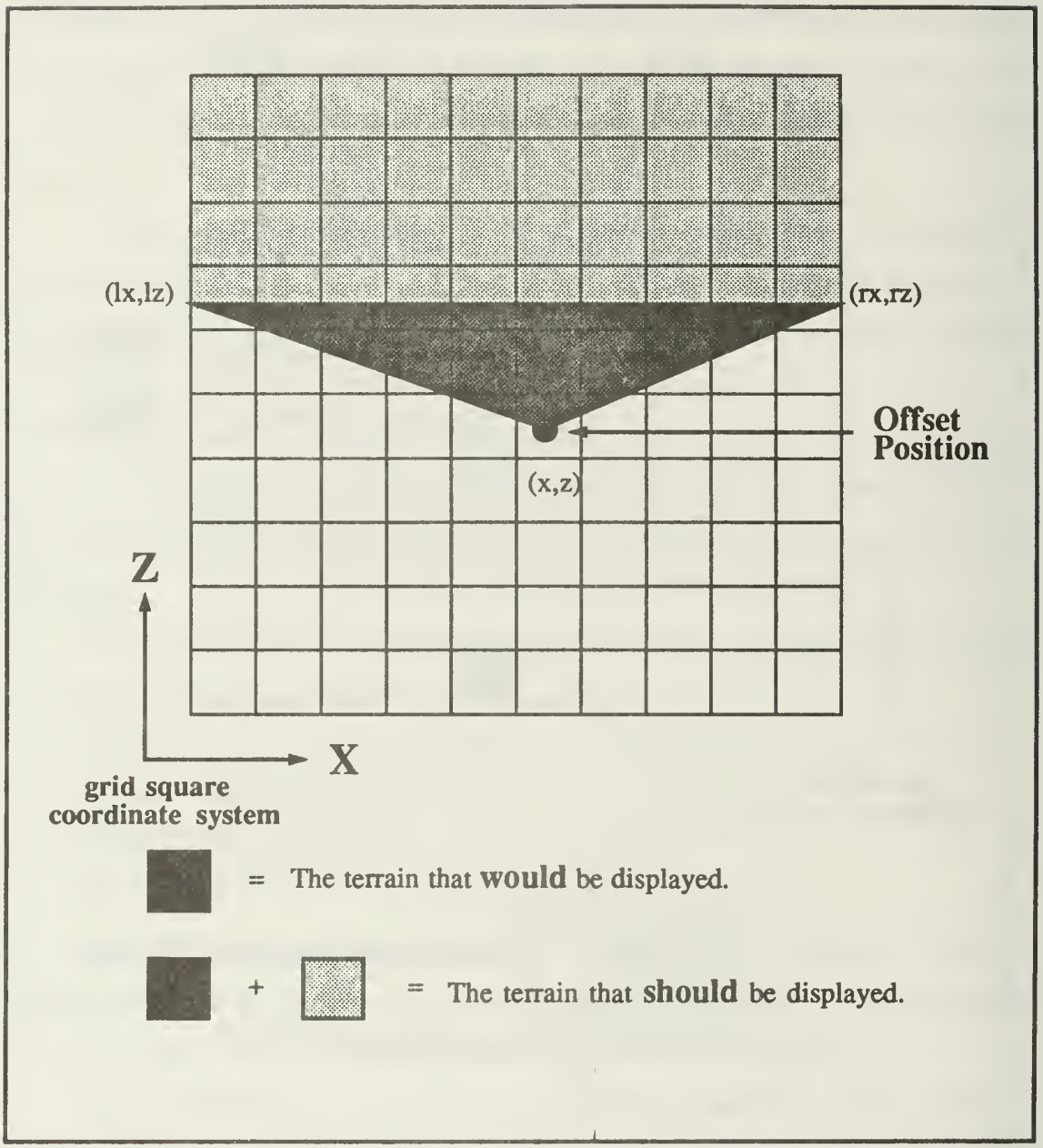

Figure 5.14 90 Degree Field-of-View Problem

```
    if (fov > 900)
{
    if ((xstart > XMIN) && (xstop < XMAX))
        if (lz < 50)
            xstart = XMIN;
        else
            xstop = XMAX;

    if ((zstart != ZMIN) && (zstop != ZMAX))
        if (lx < 50)
            zstop = ZMAX;
        else
            zstart = ZMIN;
}
```

Figure 5.15 Code to Correct 90 Degree Problem

the look angle is greater than 225 degrees and less than 315 degrees, the look direction is south, otherwise the direction is east.

If the look direction is north, the terrain is drawn from minimum to maximum z. If the look direction is south, the terrain is drawn from maximum to minimum z. If it is east, the terrain is drawn from minimum to maximum x. If it is west, the terrain is drawn from maximum to minimum x (see Figure 5.16).

Next, the limits for the near, mid, and far drawing groups are defined. If the detailed drawing option is selected, only the near group is used since all the 100 x 100 meter grid squares within the field of view are drawn. If the distance attenuation option is selected, the distance chosen for the near group is computed as a function of field of view. The near group is drawn to a further distance for smaller fields of view since the viewer is able to see further and more detailed terrain is desirable.

After every 100 x 100 meter grid square is drawn in the near group, the squares are grouped into 200 x 200 meter squares for the mid group. This is done for the next 2000 meters. The final group, which is drawn from the end of the mid group to the end of the map, contains grid squares grouped into 400 x 400 meter squares.

The starting x value (looking east or west) or starting z value (north or south) is adjusted to make sure the mid group starting grid is a multiple of two and the far group starting grid is a multiple of four. This is done so the vertices of the first row or column of each group will line up with the previous row or column.

Special tests are performed when either the left or right view lines are within five degrees of zero, 90, 180, or 270 degrees. Table 5.1 shows the drawing order when looking north or south. Table 5.2 shows the drawing order when looking east or west. Functions *compute_x_bounds*() and *compute_z_bounds*() in MPS compute these limits.

All angle measurements in Tables 5.1 and 5.2 are referenced from the x-axis, positive counter-clockwise. Quadrant one is between zero and 90 degrees, quadrant

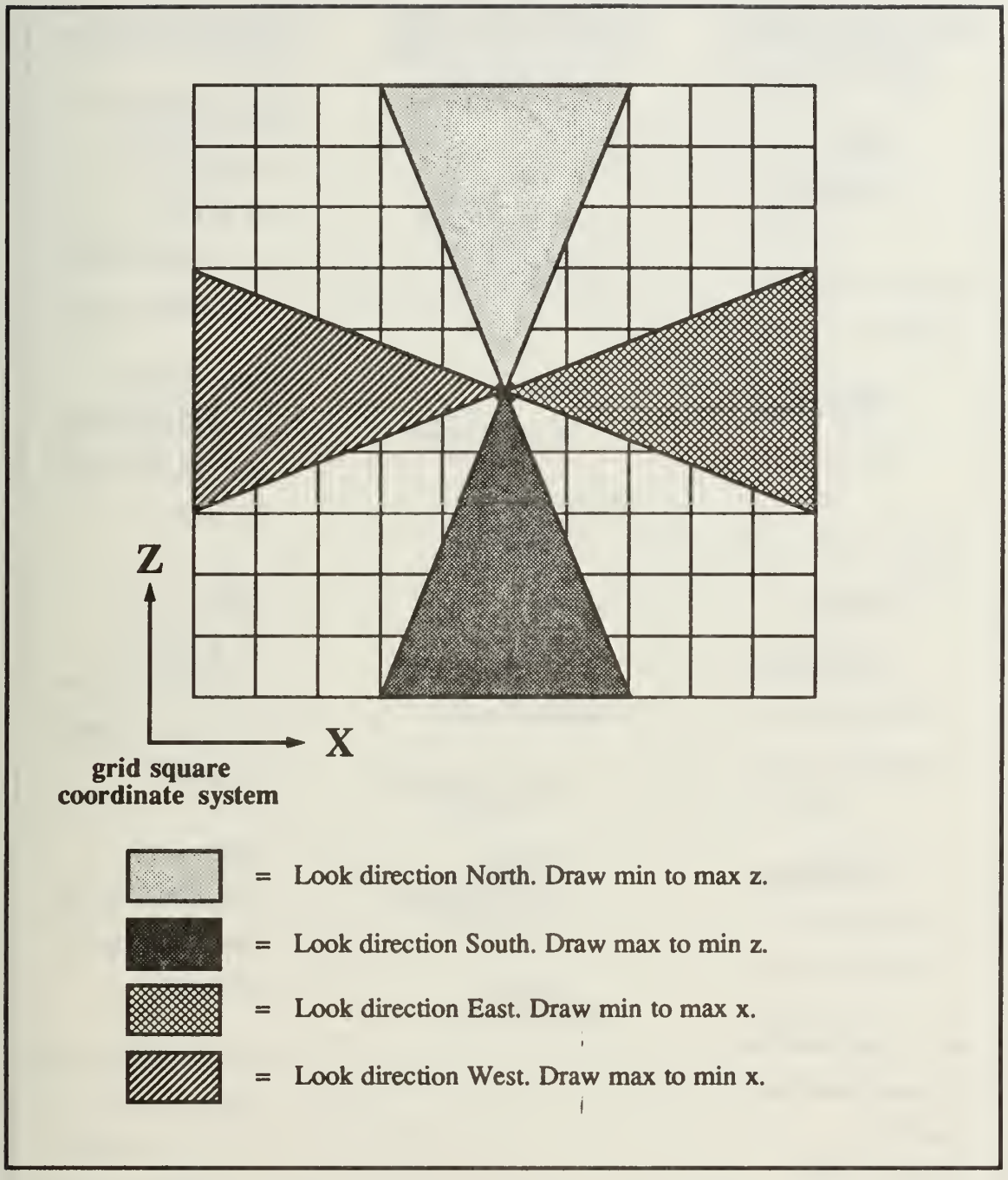

Figure 5.16 Terrain Drawing Direction

TABLE 5.1 LOOKING NORTH OR SOUTH

LEFT VIEW ANGLE OR QUADRANT	RIGHT VIEW ANGLE OR QUADRANT	DRAWING ORDER
90 deg	any	RL to ZMAX
1st quad	270 deg	ZMIN to LL
	1st or 4th quad	RL to LL
	3rd quad	if < x ZMIN to RL
		else ZMIN to LL
2nd quad	90 deg	LL to ZMAX
	1st or 4th quads	if < x LL to ZMAX
		else RL to ZMAX
	2nd quad	LL to RL
270 deg	any	ZMIN to RL
3rd quad	90 deg	LL to ZMAX
	1st quad	if < x LL to ZMAX
		else RL to ZMAX
	2nd or 3rd quads	LL to RL
4th quad	270 deg	ZMIN to LL
	2nd or 3rd quads	if < x ZMIN to RL
		else ZMIN to LL
	4th quad	RL to LL

RL = right view line
LL = left view line
ZMIN = 0
ZMAX = 100

TABLE 5.2 LOOKING EAST OR WEST

LEFT VIEW ANGLE OR QUADRANT	RIGHT VIEW ANGLE OR QUADRANT	DRAWING ORDER
0 deg	any	RL to XMAX
1st quad	0 deg	LL to XMAX
	1st quad	LL to RL
	3rd or 4th quads	if < z RL to XMAX
		else LL to XMAX
180 deg	any	XMIN to RL
2nd quad	0 deg	LL to XMAX
	1st or 2nd quads	LL to RL
	4th quad	if < z RL to XMAX
		else LL to XMAX
3rd quad	180 deg	XMIN to LL
	1st or 2nd quads	if < z XMIN to LL
		else XMIN to RL
	3rd quad	RL to LL
4th quad	180 deg	XMIN to LL
	2nd quad	if < z XMIN to LL
		else XMIN to RL
	3rd or 4th quads	RL to LL

RL = right view line
LL = left view line
XMIN = 0
XMAX = 100

two is between 90 and 180 degrees. Quadrant three is between 180 and 270 degrees, and quadrant four is between 270 and 360 degrees.

The final step in displaying the terrain involves defining the correct color for the grid squares and the correct vertices for the drawing routine. When drawing the squares in the near group, vertices of each 100 x 100 meter square along with its grid color are used. To maintain the checkerboard appearance when drawing groups of more than one grid square, first the left grid color is used followed by the right grid color of two corresponding rows or columns. The initial color for each row or column must be chosen to be different than the grid square next to it in the previous row or column. If the previous row or column used the left grid square color, then the current row or column uses the right grid square color. The opposite is true if the previous row or column used the right grid square color. An example of how the terrain is divided into the three groups is shown in Figure 5.17.

Since the smaller grid squares are grouped to form larger ones, the correct vertex coordinates of each square must be sent to the drawing routines to display the larger squares. This is described in detail in Appendix E.

When displaying the terrain using the distance attenuation option, holes can appear where the groups meet. This is shown in Figure 5.18 and is caused by grouping the grid squares for display. By filling the holes with triangles of the same material type and polygon normal vector as the adjacent grid square, the holes disappear as shown in Figure 5.19.

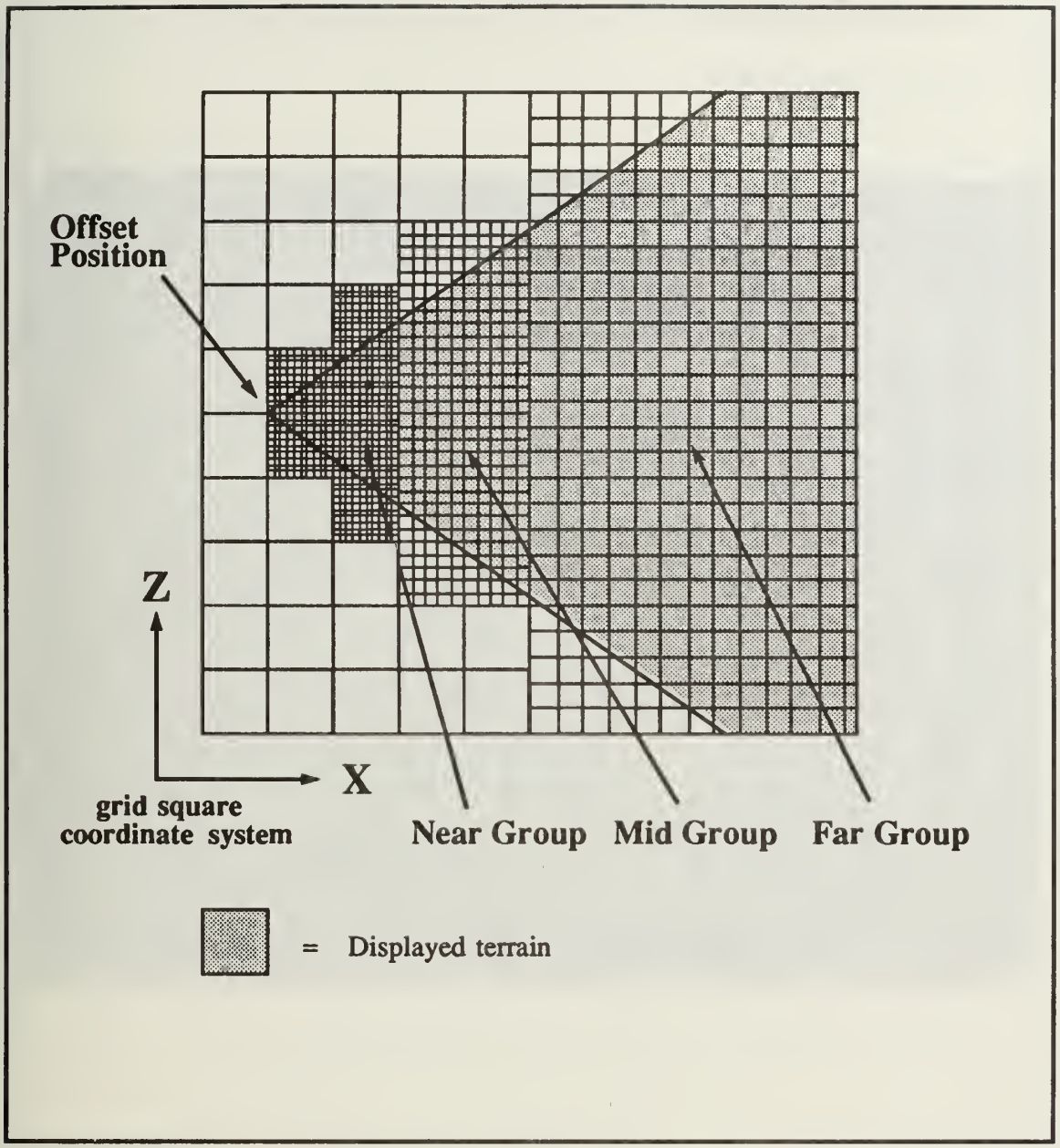

Figure 5.17 Terrain Display Attenuation Example

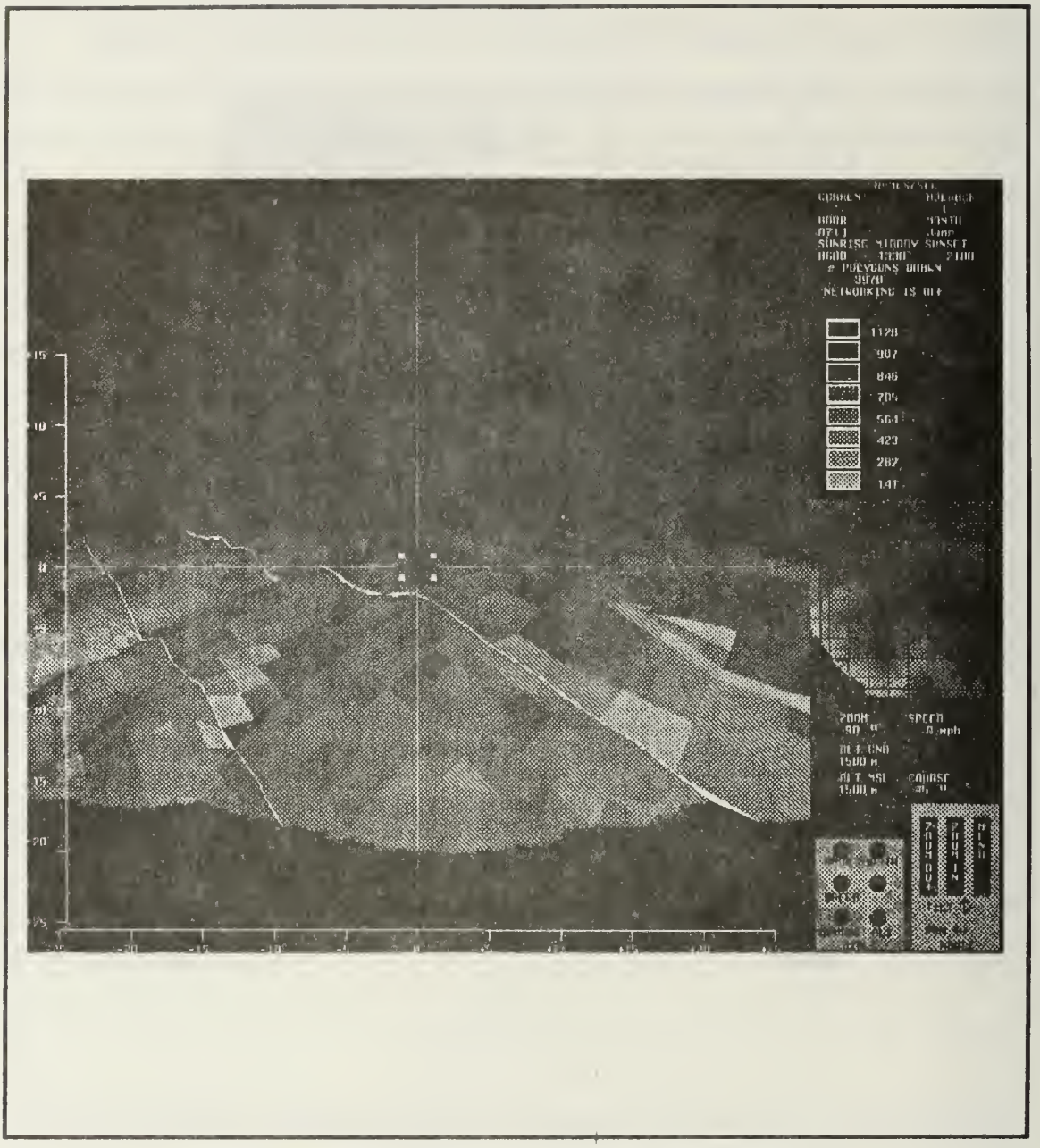

Figure 5.18 Holes from Drawing Terrain

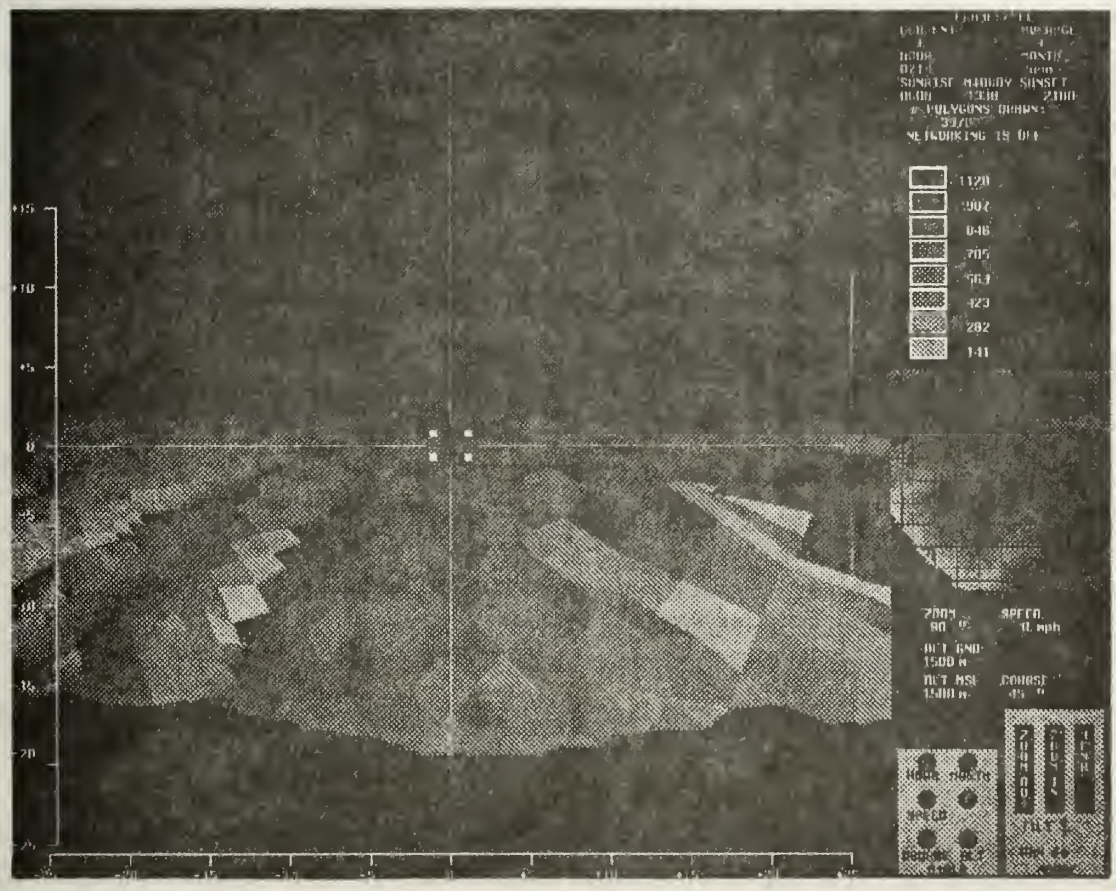

Figure 5.19 Filling Holes in Terrain

VI. NETWORKING CAPABILITY

A. OVERVIEW[2]

Networking between two or more IRIS graphics workstations requires two distinct communication levels: the first is between the machines themselves and the second is between the MPS process and the networking hardware/software. The first level is achieved by creating a separate receive process that is executed by each MPS process that is networking. This receive process contains an infinite loop that constantly monitors the network looking for packets with the correct address information. When an acceptable packet arrives, a copy is placed into an area of shared memory that facilitates the second level of communication.

The shared memory area is 1024 bytes long and is the only method of communicating between the MPS process and the receive process. By using the two levels of communications, each MPS process can send messages informing other MPS processes of significant events and can receive messages from other MPS processes.

B. MESSAGES VERSUS PACKETS

For the purpose of this study, the following definitions of the terms message and packet apply:

- *MESSAGE*: A message is the concept of needing to inform all other networked processes of an event. An example of an event is when a user changes the speed of the platform he is driving. All other processes need to know the new speed in order to correctly display that platform in their simulation.

---

[2]The files that contain all the networking functions are network.c, check_for_packets.c, and network_receive.c. Network.c and check_for_packets.c contain all functions used by the MPS process and network_receive.c is the receive process that constantly monitors the network. All these files use the header file network.h.

- ***PACKET***: A packet is a highly structured collection of characters, consisting of two parts: **header** and **body**. The header contains all the information necessary to identify the **type** of packet, and the body contains all the **data fields** that are required for that type of packet.

C. MESSAGES

1. Generation Methodology

Knowing when to generate a message was one of the biggest problems in designing the networking system. The following events were chosen to be the ones that require a message to be sent:

- Send an **init** message when initially entering network mode
- Send an **answer** message when responding to an init message
- Send one **update** message for every local platform whenever any one of two events occur: The user has selected (from the main menu) a platform to operate, or the user has changed all the platforms' speeds from one of the operating menus
- Send an **end** message when quitting the simulation, or after adding or deleting any or all platforms
- Send a **zero velocity** message whenever an operating menu is displayed, so that all the local platforms stop moving on all other simulations, and then send a **normal velocity** message when finished selecting from the menu
- Send an **update** message for the platform that is being driven whenever the user makes a change to the course, speed, or altitude
- Send a **lock on** message after locking onto a platform that is from a non-local simulation, and send a **lock off** message when disengaging from it
- Send a **destroy** message if a local FOGM missile has destroyed a non-local platform
- Send a **crash** message if a non-local platform has crashed into any other platform, wreck or obstacle

2. Types

After we decided what events needed to generate messages, the formal types of messages needed to be defined. The header file, network.h, defines the different types of messages that are available as follows:

- INITIALIZATION
- ANSWER
- UPDATE
- ALL
- ZERO_VELOCITY
- NORMAL_VELOCITY
- END
- LOCK_ON
- LOCK_OFF
- DESTROY
- CRASH

The **initialization** message is sent when the simulation wants to find out if any other MPS processes are currently networking. The only information that must be sent is the fact that it is an initialization message.

The **answer** message is sent in response to an initialization message. Three pieces of information need to be sent as follows:

- The local simulation's base identification number. This number is the beginning of a block of 10,000 integers that are used to uniquely identify the platforms from this simulation. The non-local simulation will add 10,000 to this value to get its base identification number
- The x_grid for the operating area
- The y_grid for the operating area

The **update** message is specifically about the platform that is being driven. This message is generated when the course, speed or altitude of a platform is changed and must contain the following information:

- The network identification number of the platform
- The platform type code
- The x location of the platform
- The z location of the platform
- The X rotation of the platform (tilt)
- The Y rotation of the platform (ang)

- The Z rotation of the platform (inc)
- The velocity of the platform
- The altitude of the platform
- The course in degrees of the platform

The **all** message must contain the same information as the update message, however there must be a message for every platform in the local simulation.

The **zero velocity** message is similar to the **all** message except that the velocity field is changed to zero. The **normal velocity** message is identical but the correct value for the velocity is sent instead of zero.

The **end** message has only one field which contains the base_id_number for the simulation.

The **lock on** and **lock off** packets are very similar. The only information that must be sent is the tracked_net_veh_id of the platform that was just locked onto or released.

The **destroy** and **crash** messages also contains only one field. This field contains the destroyed_net_veh_id or the crashed_net_veh_id of the destroyed platform.

D. PACKETS

As previously stated, a packet is a highly structured collection of characters, consisting of two parts: **header** and **body**. The header contains all the information necessary to identify the **type** of packet, and the body contains all the **data fields** that are required for that type of packet.

1. Format

A packet is transmitted as a sequence of characters (a string). Therefore, any information that must be sent that is not of type character (such as integer, short, float, or double) must be converted to a string before it can be inserted into a packet. The system function *sprintf*() is used to convert integers and shorts to a string. For numbers of type float and double, the number must first be converted to an integer

type. Since doing a straight conversion to integer (truncation) would destroy everything after the decimal point, each float or double is multiplied by a constant called PACKING_FACTOR. The value for the PACKING_FACTOR is located in the header file network.h and is currently 10,000. When a packet is received, the PACKING_FACTOR is divided out of the appropriate fields of the packet.

a. Header

The header for every packet is fixed at 50 characters long. The first four characters repeat a unique token defined for the particular packet type. The different tokens are as follows:

- Initialization packet token *
- Answer packet token #
- Update packet token @
- End packet token $
- Lock on packet token %
- Lock off packet token ^
- Destroy packet token !
- Crash packet token &

The middle part of the packet is a human readable description of the packet owner and type. For example: "MOVING PLATFORM SIMULATOR destruct packet." The only change in this part of the header from one packet to another is the word that describes the type. The last part of the header is a repeat of the unique packet token. Enough of these tokens are placed at the end of the header to pack it to 50 characters long. All the legal headers are listed in Figure 6.1.

b. Body

The body of a packet is strictly defined for each type. Table 6.1 outlines these definitions.

"**** MOVING PLATFORM SIMULATOR initial packet ****"

"#### MOVING PLATFORM SIMULATOR answer packet #####"

"@@@@ MOVING PLATFORM SIMULATOR update packet @@@@@"

"$$$$ MOVING PLATFORM SIMULATOR end packet $$$$$$$$"

"!!!! MOVING PLATFORM SIMULATOR destruct packet !!!"

"%%%% MOVING PLATFORM SIMULATOR lock on packet %%%%"

"^^^^ MOVING PLATFORM SIMULATOR lock on packet ^^^^"

"&&&& MOVING PLATFORM SIMULATOR crash packet &&&&&&&"

Figure 6.1 Packet Headers Available to MPS

2. Examples

For the purpose of this paper only, all spaces in the following packets have been replaced by periods. Actual packets do contain spaces.

An **initialization** packet is always the same and does not contain any information fields as follows::

****.MOVING.PLATFORM.SIMULATOR.initial.packet.****

A typical **answer** packet contains three information fields as follows:

####.MOVING.PLATFORM.SIMULATOR.answer.packet.#####
10000.....610.....700.....

TABLE 6.1 PACKET BODY DEFINITIONS

PACKET TYPE	NUMBER OF FIELDS	FIELD CONTENTS	DATA TYPE	FIELD WIDTH bytes
INIT	0			
ANS	3	base_id_number	int	20
		x_grid	short	20
		y_grid	short	20
UPDATE	10	platform*->net_veh_id	int	20
		platform->t	short	20
		platform->x	Coord	20
		platform->z	Coord	20
		platform->tilt	short	20
		platform->ang	float	20
		platform->inc	short	20
		platform->vel	float	20
		platform->alt	float	20
		platform->cse	float	20
END	1	base_id_number	int	20
LOCK_ON	1	tracked_net_veh_id	int	20
LOCK_OFF	1	tracked_net_veh_id	int	20
DESTROY	1	destroyed_net_veh_id	int	20
CRASH	1	crashed_net_veh_id	int	20

*Platform is a pointer to the Vehicle structure of the platform concerned.

An **update** packet is the most complex packet because it contains ten information fields as follows:

@@@@.MOVING.PLATFORM.SIMULATOR.update.packet.@@@@@
10001.....3.....29098518.....38966032.....10.....20943
.....-10.....690400.....5895714.....3300000.....

A typical **end** packet contains one information field as follows:

$$$$.MOVING.PLATFORM.SIMULATOR.end.packet.$$$$$$$$10000.....

Lock_on, lock_off, destroy, and **crash** packets each contain one information field as follows:

%%%%.MOVING.PLATFORM.SIMULATOR.lock.on.packet.%%%%10001.....

^^^^.MOVING.PLATFORM.SIMULATOR.lock.off.packet.^^^10001.....

!!!!.MOVING.PLATFORM.SIMULATOR.destroy.packet.!!!10001.....

&&&&.MOVING.PLATFORM .SIMULATOR.crash.packet.&&&&&&
10001.....

E. SHARED MEMORY

UNIX System 5.0 has system commands that allow multiple processes to share a common block of memory. Through the use of this capability, the networking of MPS processes is possible. The file network.c contains the functions that create the shared memory area and attach to it. The file network_receive.c contains the separate receive process that attaches to the same shared memory segment. The particulars of the content of the file network_receive.c are discussed later.

If networking is selected by the user, MPS calls the function *network*() as follows:

network(NETWORK_SETUP)

This function call causes the function *shared_memory_semaphores_and_network_setup*() to be called which calls the system function *shmget*() as follows:

shmget(SHARED_MEMORY_KEY, SHARED_MEMORY_SIZE,
0777|IPC_CREATE)

The manifest constants SHARED_MEMORY_KEY and SHARED_MEMORY_SIZE are defined in the header file network.h which is part of the MPS system. IPC_CREATE is defined in the system header file **sys/shm.h**. This call to *shmget*() returns the shared memory identification number for the newly created shared memory segment.

The next system function that is called is *shmat*(). This function returns a pointer to the first memory location of the shared memory segment. This address is used to read and write information out and in to the shared memory segment. The last operation that the function *shared_memory_semaphores_and_network_setup*() performs in relation to shared memory is to clear all the memory locations to zero.

F. NETWORK INITIALIZATION

The physical interconnection between different IRIS graphics workstations in the Graphics and Video Laboratory is through an Ethernet local area network. The function *shared_memory_semaphores_and_network_setup*() initializes the network interface by calling the function *getbroadcast*() which is located in the same file.

The first function that *getbroadcast*() calls is *getservbyname*() which is a system function that looks in the system file **/etc/services** for the line of code in Figure 6.2. If this line of code is present, then a pointer to a structure containing the line of code is returned. If this line of code is not present, then interprocess communication can not

```
udpbrdcst    2000/udp    broadcast
```

Figure 6.2 Line of Code That Must Appear in the System File /etc/services

occur. Therefore to implement networking on an IRIS workstation, a super user must add the line to the file **/etc/services.**

The next function called is *socket*(). This function returns a value of type **int** that is a descriptor or number of the socket that will transmit and receive the packets for MPS.

Next, the function *setsockopt*() configures the socket for the type of communications that MPS needs. In particular, the socket must be set up for broadcast communications.

The function *bzero*() is called to clear the structure containing the socket addressing information. Then the correct address information is placed in the structure by three assignment statements.

Finally, the function *ioctl*() puts the socket into non-blocking mode so that the process looking at the socket for packets does not have to wait if no packet is present. This prevents the process from becoming blocked.

G. RECEIVE PROCESS

The receive process is constantly monitoring the Ethernet looking for packets with the correct header information. When a correct packet arrives, its contents are placed into a local string called message that is as large as the shared memory segment. Then the message is copied into the shared memory segment by the *strcpy*() function. The algorithm for the receive process is outlined in Figure 6.3.

```
while(TRUE)
{
        wait_for_a_packet(message);

        /* Put the received message into the shared memory area */
        strcpy(address,message);
}
```

Figure 6.3 Algorithm For Receive Process

VII. PERFORMANCE EVALUATION OF MPS

One objective of our research is to evaluate the performance of the Moving Platform Simulator on the IRIS 4D/70GT, a high performance graphics workstation. Measurements were made during the main event loop while the platforms and terrain were being displayed in the map window. First let us review the complexity of the drawing being displayed. The process includes:

- Z-buffering
- infinite light source
- equations to update each platform's position
- equations to update each platform's grid position, insertion, and deletion in the appropriate linked list
- computations of the look direction
- computations of the left and right view angles, total field of view area, and offset position
- computations to determine which grid squares to display, and where the boundaries occur between the near, mid, and far drawing groups
- computations for collision detection
- activation of the perspective view of the world
- activation of the lighting model
- the display of each polygon of the terrain and platforms in view
- the display of the updated indicator information in the menu, navigation, and indicator windows

The Moving Platform Simulator is totally different from any of the previous 3D simulators developed in the Graphics and Video Laboratory at the Naval Postgraduate School. Many new options and capabilities are incorporated in this simulation, most of these due to the additional capabilities of the IRIS 4D/70GT.

Some new options degrade performance. The most noticeable of these is the decision to draw the terrain all the way from the offset position to the edge of the map within the field of view lines. This is in contrast to drawing just 2000 meters from the driven position as in earlier simulations. Although the terrain is grouped and drawn

for distance attenuation, performance is decreased since more terrain is drawn upon each update.

A. BENCHMARK PERFORMANCE

The MPS simulator is much more sophisticated than either the VEH or VEH II simulators, and cannot be considered in the same group. However for completeness, two of the original tests were performed on MPS to measure its performance. (See Chapter II and Tables 2.1, 2.2, and 2.3 for the results of the tests on the older simulators) The first test involved one vehicle on the terrain and the second test involved eight vehicles within the field of view of a jeep. Table 7.1 shows the performance for MPS for these tests on the IRIS 4D/70GT. The results of these tests show that MPS is running at about the same speed as VEH II does on the IRIS 4D/70G. However, the pictures are better and the capabilities are greater in the Moving Platform Simulator.

Since MPS is a new simulator, new benchmarks are needed to measure its performance on graphics workstations. Quantities such as polygons per frame and frames per second are important in measuring the performance of visual simulations.

The case chosen to conduct the performance measurements of MPS is one in which a single missile is operated with as much terrain as possible in the field of view. The missile is flown at its maximum altitude of 1500 meters above the grid square (5,5) looking northeast. The view from the missile looking down on the terrain from a high altitude allows the terrain grid squares to be drawn with a somewhat consistent polygon size. Both the detailed and distance attenuation drawing options for 90 degree and 10 degree field of views were analyzed. Table 7.1 also shows the results from these cases using an IRIS 4D/70GT. These performance measurement values should be used as a benchmark for future evaluations.

TABLE 7.1 MPS PERFORMANCE MEASUREMENTS

DISPLAYING DETAILED TERRAIN

PLATFORM	ZOOM ANGLE	POLYGONS PER FRAME	FRAMES PER SECOND
ONE VEHICLE	55	763	8
ONE VEHICLE	15	403	14
NINE VEHICLES	55	1086	6
NINE VEHICLES	15	722	8
MISSILE 1500m	90	19801	< 1
MISSILE 1500m	10	3387	2

DISPLAYING ATTENUATED TERRAIN

PLATFORM	ZOOM ANGLE	POLYGONS PER FRAME	FRAMES PER SECOND
ONE VEHICLE	55	607	9
ONE VEHICLE	15	393	15
NINE VEHICLES	55	940	7
NINE VEHICLES	15	680	9
MISSILE 1500m	90	4152	2
MISSILE 1500m	10	816	7

VIII. SYSTEM EVALUATION OF MPS

A. SYSTEM LIMITATIONS

1. Networking

Whenever either process wants to access the shared memory area, no check is made to insure that the other process is not also accessing the area. Potentially, this problem can cause packets to be lost or corrupted. A logical improvement to MPS would be to add semaphores to control access to the shared memory area.

2. Collision Detection

The current implementation of collision detection is very time intensive and inefficient. The algorithm that is used compares the first platform with all the rest, the second platform with all but the first, the third platform with all but the first two, etc. A logical enhancement to MPS would be to improve this algorithm so it is more intelligent and faster.

3. Terrain Display

The terrain that is displayed within the field of view is divided into three groups. The near group, which extends from the offset driven position and is a function of the field of view, contains each 100 x 100 meter squares. The mid group begins where the near group ends and extends for 2000 meters. Here the squares are displayed with four 100 x 100 meter squares drawn as one square.

The breakpoints for the groups are determined by the cosine of the look direction and the distance desired. For example the stopping point for the near group can be given by:

```
xstop[NEAR] = xstart[NEAR] + cos(lookang) * num_poly_near
```

The equation works well when the look direction is exactly 0, 90, 180, or 270 degrees since only complete columns are drawn when looking east or west, and complete rows are drawn when looking north or south. However, if the look direction is 45 degrees for example, more terrain is drawn within the group than desired. Figure 8.1 shows what terrain should be drawn in each group, while Figure 8.2 shows what is actually drawn.

This limitation in the display algorithm means that the performance can be increased if the terrain can be grouped for display as shown in Figure 8.1. However problems arise if complete rows or columns are not displayed in the same group. The algorithm would have to be modified to track where the breakpoint occurs within the group. Also, **holes** which are now seen and corrected on group boundaries would occur within the group. We feel the extra effort to display the terrain as given in Figure 8.1 would further complicate the algorithm, and the benefits do not justify the expense.

Another limitation with the display algorithm is the representation of the vehicles on the larger grid squares. Recall each vehicle is drawn on the terrain based on the grid square's orientation beneath it. If the terrain is inclined so is the vehicle to give the appearance the vehicle is moving over the terrain. When the 100 x 100 grid squares are grouped to form larger squares for display, the characteristics of the terrain change as the large square is drawn. The vehicles, however, are still drawn as they would have appeared in their original grid square.

Correcting this limitation would involve recomputing the vehicles orientation based on the larger grid square. Since the vehicles in this group are at least 2000 meters from the driven platform, it was decided to draw the vehicles correctly was not worth the extra effort needed.

The final terrain limitation concerns the color chosen for the larger grid squares. When the four 100 x 100 grid squares are grouped and drawn as one square, four possible colors exist for that square. To simplify the choices the color of the first

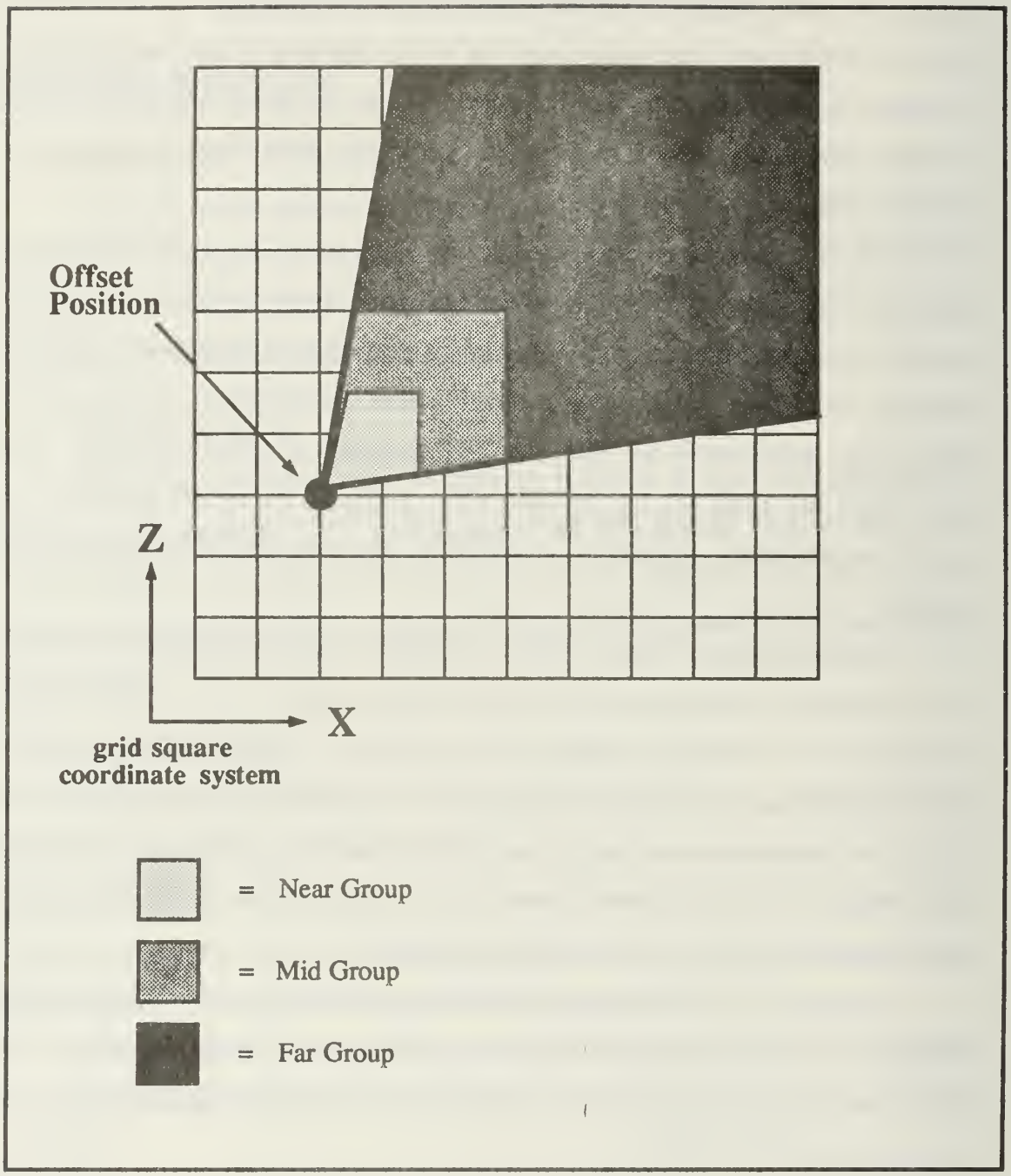

Figure 8.1 How the Terrain Should be Displayed

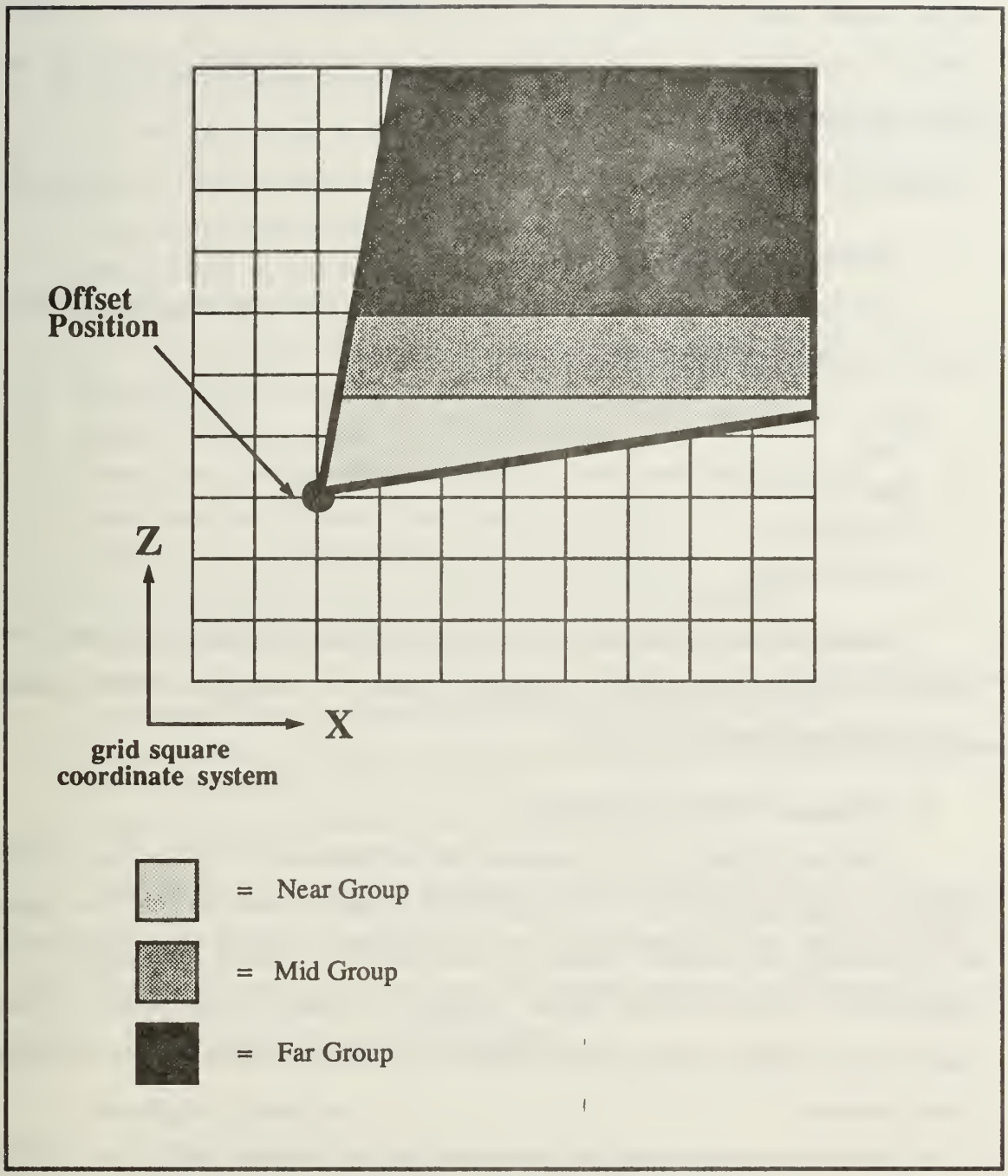

Figure 8.2 How the Terrain is Displayed

100 x 100 grid square is used for the entire larger square, unless the checkerboard effect is violated as described earlier. If this occurs, the color of the 100 x 100 grid square directly next to the first one is then used. This means some colors may be seen in the small 2D map in the navigation window but not drawn in the terrain display of the map window.

B. FUTURE CAPABILITIES

1. Additional Platform Types

Currently there are five platform types available in the Moving Platform Simulator. These platforms are as follows:

- Tank
- Truck
- Open jeep
- Covered jeep
- FOGM missile

Adding additional platforms is not necessarily an easy task since many routines would need to be created or modified. Appendix D outlines the steps necessary to add additional platforms to MPS.

2. Enhanced Lighting and Shading

Real-time lighting of the platforms was implemented in the Moving Platform Simulator. Material types and color values were chosen to make the platforms appear as realistically as possible, however, more work is needed to improve their appearance. Currently all the outside polygons of each platform have the same material type. Perhaps shading certain portions of platforms differently would improve their appearance.

No work was done exploring the capabilities of the alpha buffer of the lighting model. This could be used to give a translucent effect for the missile's smoke trail or wreck's flame. Finally, fake shadows could be added using scaling and half tone grey

polygons. These improvements must be weighed against the decrease in execution time to determine if they are feasible.

3. Realistic Platform Dynamics

No attempt is made to simulate real-world moving platform dynamics in MPS. Dynamics that could be added to the simulation, however include the following items:

- Human forms inside platforms that appear to be operating them
- Speed limitations, both forward and backward, for ground platforms
- Minimum speed (or fixed speed) of the FOGM missile
- Cab vibration and the bumpy way platforms actually behave
- Maximum grades that a platform can travel up and down
- Platform "tip over" angles
- Braking effects that could take into account momentum, inertia, etc.
- Brake lights that illuminate when brakes are applied
- Headlights that actually light the terrain in front of them
- Neutral steer capability of tanks (ability to turn while stationary)
- Realistic steering (instead of only a direction control that responds instantly)
- Realistic collisions (a tank would destroy a jeep but not be destroyed itself, and two tanks colliding would randomly destroy each other or both, etc.)
- Realistic interiors of ground platforms (steering wheels, dashboards, instrument clusters, seats, pedals, etc.
- Windows that can be seen with reflections, smudges, etc.

Adding any or all of these dynamics would decrease the speed of the simulation which is the primary reason they are not implemented on this version of MPS. If these improvements are made, they must be weighed against the decrease in execution time to determine if they are actually desired.

4. Intelligent Platforms

No attempt is made to integrate the Moving Platform Simulator with any type of Artificial Intelligence (AI) machine. However, since networking mode allows any machine connected to the Ethernet to receive the broadcast packets from MPS, controlling a platform from another machine is not difficult. An expert system that can

make decisions about the speed and direction of a platform is already under consideration [Ref. 11].

5. Dynamic Operating Area

The original versions of the FOGM and VEH simulators restricted the operating area to 10x10 kilometers. This restriction has proliferated through all subsequent versions of VEH, VEH II, and MPS. If this restriction were removed, users would not have to select a 10x10 kilometer area but would be able to place platforms anywhere on the 35x35 kilometer map. This would simplify the user interface and make the simulation a little more streamlined.

6. Use of Defense Mapping Agency (DMA) Type II Digital Terrain Elevation Data (DTED)

The terrain database that MPS currently uses is not in a standard format. Most digital terrain elevation databases available in the military are in the standard DMA type II format. Modifying the routines that read the elevation data would enable MPS to accept data for almost anywhere in the world.

C. FUTURE MACHINES

The Moving Platform Simulator was developed on a IRIS 4D/70GT workstation, which was Silicon Graphics' most advanced workstation at the time of program development. Now specifications for even more advanced workstations are available. These units will become available in the near future and should increase the capabilities of simulations like MPS.

1. SGI IRIS 4D/70GTX

The IRIS 4D/70GTX system contains hardware arranged in a new architectural design that should increase the computing power of the system extensively. This unit contains two 16.7 MHz RISC processors along with two floating point coprocessors. Disk capacity is anywhere between 380 MB to 9.6 GB of space. Any

IRIS 4D system can be upgraded to the 70GTX version with minimal changes [Ref. 12:p. 1].

The 70GTX is rated at 20 MIPS and should be capable of producing 100,000 Z-buffered four-sided, Gouraud shaded, Phong lighted polygons per second. The additional processor and new architectural design will improve system performance [Ref. 12:p. 2].

2. SGI IRIS 4D/240GTX

In the near future Silicon Graphics should begin delivery of their next generation of workstations. The 240GTX contains four CPUs, each operating at 25 MHz. The power of the new processors should provide increased performance in both technical computations and graphics processing [Ref. 13].

The 240GTX should be fully compatible with applications previously developed on IRIS GT workstations. Its four processors should allow even more computations to be performed within real-time 3D simulations like MPS. Currently our mobility expert system is forced to be executed on another processor and interfaced over a network. Perhaps using a 240GTX workstation, one processor could be used for these computations while other processors could execute the simulation system. [Ref. 13]

IX. CONCLUSIONS

This study originated from our desire to provide a meaningful graphics application as a benchmark for high performance graphics workstations. In order to accomplish this goal, we needed to extend the capabilities of our previous 3D visual simulators, such simulators being the applications paradigm. We then needed to provide full explanations of the operations involved in that simulator, both computationally and graphically.

One of the major changes from our previous simulators that we made for MPS was the utilization of Z-buffering for all hidden surface elimination. Previous simulators relied upon a scanline hidden surface elimination algorithm performed in the CPU. The scanline algorithm greatly complicated the simulator's software and made it less supportable over the long run. It turns out that selecting Z-buffering for all hidden surface elimination at this time is fortuitous as Z-buffers have just become fast enough to beat our older scanline algorithm.

An additional benefit of utilizing Z-buffering is that we are now exhausting a graphics capability and getting meaningful numbers of our own that we can now compare to numbers the manufacturers are citing. This alone is a significant addition to the graphics workstation benchmarking literature.

Another change we made to our previous simulators was to use the newly available lighting and Gouraud shading capabilities of our workstations. We did this by adding into our system an ad-hoc, somewhat realistic model for the sun and its movement during the day. We did this because on the IRIS 4D/70GT an infinite light source with an infinite viewer is free timewise when Z-buffering is turned on.

Overall we have used our 3D visual simulator to push the IRIS 4D/70GT to its limit. It performs acceptably but we certainly want it faster. We expect that to happen with soon to be available hardware. When that faster hardware arrives, we will again

benchmark it with MPS with the hope that perhaps what we do will provide a more meaningful graphics workstation comparison than what has been previously been available.

APPENDIX A USER INTERFACE

The user interface of any application program must be designed so that novice and experienced users alike can effectively operate the program with little or no help from user's manuals or other users. A thorough and efficient design of command line options, popup menus, dials, and a mouse achieves this.

A. COMMAND LINE OPTIONS[3]

MPS currently has three options available from the command line.

- Network mode
- Test mode
- Silent mode

Selection of the network mode activates the networking capabilities of the program. If one or more MPS processes are operating on different machines, then they will be able to share information regarding the other platforms. When a platform changes course, speed or altitude (FOGM only), a broadcast packet is sent to all other MPS processes and the appropriate platform's information is updated.

Selection of test mode bypasses some of the cosmetic portions of the program. Currently, the only part that is bypassed is the opening billboard sequence.

Selection of silent mode turns off the bell that rings to indicate acceptance of input from the user. This option is useful for demonstrations when the ringing would interfere with a verbal explanation of the program.

---

[3]The code that processes the command line arguments is contained in the file decode_arguments.c.

B. POPUP MENU SYSTEM[4]

Popup menus are the primary source of user input into the program. There are currently 24 different popup menus that are used in various parts of the simulation. If a selection in a menu is not allowed or meaningful when the menu is displayed, the selection is displayed in lower case. Otherwise the selection is completely uppercase. We did not omit disallowed selections so that the menus always appear in the same order and format every time. If we were to eliminate disallowed selections, users would tend to be overwhelmed by the number of different menus. In fact, of the 24 menus in the system, only 13 are really unique. A detailed explanation of each menu follows:

1. Opening Menus

There are two menus that make up the opening menu set. These menus are called OPENING_ONE and OPENING_TWO. Each of these menus contain the same four selections as follows:

- GO TO NEXT INTRODUCTION PAGE
- GO TO SELECT AREA MENU
- EXIT THE PROGRAM
- ENTER 4SIGHT (RESIZE OPTIONS)

OPENING_ONE allows the user to select any one of these options but OPENING_TWO disallows the first option. OPENING_TWO is displayed if the user is currently looking at the last introduction page.

---

[4]The code for defining all popup menus is contained in the file makepopups.c. Code for displaying and processing menu selections is contained in the following files: do_main.c, do_main_1.c, do_main_2.c, do_main_3.c, do_main_4.c, do_driving_menu.c, do_flying_menu.c, do_change_speed.c, do_intros.c, do_quitting.c, do_select_area.c, do_the_add.c, do_the_defaults.c, do_the_delete.c, and do_the_select.c.

The first selection allows the user to toggle through the predefined set of introduction screens. These screens give some history behind the evolution of the simulator and give credit to those individuals and organizations that have contributed to its development.

If the last introduction page is being displayed or the user wishes to bypass the introduction pages, the GO TO SELECT AREA MENU selection will do just that. To exit the program, the user must select EXIT THE PROGRAM and a small menu will be displayed with the following selections:

- RETURN TO WHERE YOU WERE
- REALLY QUIT

If the user desires to resize or move the simulation's windows, the option ENTER 4SIGHT (RESIZE OPTIONS) will allow him to accomplish it. After selecting the option, the windows will be cleared to white and the user can click on the menu bar and move or resize as desired.

2. Select Area Menus

There are two menus that make up the select area menu set. These menus are called SELECT_AREA_ONE and SELECT_AREA_TWO. Each of these menus contain the same eleven selections as follows:

- SELECT AN AREA OF THE MAP
- GO TO MAIN MENU
- EXIT THE PROGRAM
- ENTER 4SIGHT (RESIZE OPTIONS)
- COLOR SCHEME - BROWN RAMP
- COLOR SCHEME - MULTIPLE COLORS
- COLOR SCHEME - GREY RAMP
- COLOR SCHEME - RED RAMP
- COLOR SCHEME - GREEN RAMP
- COLOR SCHEME - BLUE RAMP
- GO TO INTRODUCTION SCREEN

SELECT_AREA_ONE allows the user to select any one of these options but SELECT_AREA_TWO disallows the first option. SELECT_AREA_TWO is displayed if the user is not the first simulator to select network mode. If this user were to select a different area to operate in while networking, there would be no correlation between platforms from other processes and the terrain area the user was operating in. This is why the first simulation to enter networking is allowed to select an area of the map in which to operate.

Selecting GO TO MAIN MENU will take the user to the main menu which is the next logical place to go after selecting a location to operate.

The color scheme selections change the way the terrain is colored. Each color scheme has eight different colors that are based on the elevation at that location. The simulation actually uses 16 colors to create a checkerboarding effect, however the user is only shown the eight primary colors in the color ramp.

The last selection allows a user to return to the introduction screens if he desires.

3. Main Menus

There are four menus that make up the main menu set. These menus are called MAIN_ONE, MAIN_TWO, MAIN_THREE, and MAIN_FOUR. Each of these menus contain the same eight selections as follows:

- PLACE DEFAULT SET OF PLATFORMS
- ADD A PLATFORM
- DELETE A PLATFORM
- SELECT A PLATFORM TO OPERATE
- EXIT THE PROGRAM
- ENTER 4SIGHT (RESIZE OPTIONS)
- SAVE PLATFORMS TO A FILE
- SELECT ANOTHER AREA OF THE MAP

MAIN_ONE is the first menu that is displayed after selecting an area of the map. Since there are no platforms displayed at this point, the delete, select and save options are disallowed. After adding as few as one platform, MAIN_TWO is displayed which allows all selections on the menu. MAIN_THREE is displayed only when the act of adding default sets of platforms would exceed an arbitrary limit on the number of platforms allowed in the simulation at any one time. MAIN_FOUR is displayed when the limit on the number of platforms displayed has been reached.

Selecting the first option (PLACE DEFAULT SET OF PLATFORMS) will display another menu called DEFAULT_MENU. This menu contains 6 selections as follows:

- ENTER THE FILENAME FOR YOUR PLATFORMS
- CONVOY - 10 GROUND PLATFORMS
- CONVOY - 10 GROUND & 1 FOGM PLATFORM
- JEEPS - 20 IN A ROW
- DR. ZYDA'S CONVOY
- DR. ZYDA'S WILDMAN DEFAULTS

If the user selects the first option, a small window is displayed on the screen which prompts the user for the filename. If valid information is found in the file, the appropriate platforms are added to the simulation. The main menu is then redisplayed.

Selection of any other option on the DEFAULT_MENU results in the addition of predesignated platforms in predesignated locations. These selections are useful for demonstration purposes and for persons interested in getting some platforms on the screen very quickly.

The information for the default sets of platforms is contained in data files that are read when indicated by a menu selection. The complete path for these files is contained in the header file files.h.

The next option on the main menu is ADD A PLATFORM. Selecting this option displays the following menu:

- ADD A COVERED JEEP
- ADD AN OPEN JEEP
- ADD A TRUCK
- ADD A TANK
- ADD A FOGM MISSILE
- ADD AN OBSTACLE

If a moving platform is selected (jeep, tank, truck, or FOGM), menus are displayed requesting an initial speed and direction for the platform. If an obstacle is requested then the speed and direction menus are bypassed. The FOGM missile defaults to an initial altitude of 200 meters above the terrain at the point where it is placed. After completing the selections, an icon is placed on the screen that resembles the selected platform or obstacle. The user can then move the icon with the mouse and place the platform by clicking the right mouse button. After placing the icon on the screen, the main menu is displayed once again.

Selecting the DELETE A PLATFORM option displays the following menu:

- DELETE A SINGLE PLATFORM
- DELETE ALL PLATFORMS ON THE SCREEN

If the user wants to delete one platform, an **X** cursor is displayed and the user can click on the desired platform. If the user wants to delete all the platforms on the screen, the following menu is displayed:

- NO, DO NOT DELETE ALL THE PLATFORMS
- YES, DELETE ALL PLATFORMS

The appropriate selection from this menu either cancels the operation or executes it. This menu prevents a user from deleting vehicles that he may not really want to delete.

The next selection from the main menu is SELECT A PLATFORM TO OPERATE. If the user selects this option, the following menu is displayed:

- ZOOM IN TO ANY LEGAL GRID SQUARE
- SELECT A PLATFORM TO OPERATE RIGHT NOW

The zoom option is usually necessary if platforms are close to each other and the individual icons overlap. By zooming into the 1x1 kilometer grid square, the user can more easily select the platform he desires.

If the platform the user wants to operate is clearly visible, then the second selection allows the user to select a platform immediately.

If the user has placed platforms on the screen and wishes to save them to a file, then the main menu selection SAVE PLATFORMS TO A FILE accomplishes this. A window opens that prompts the user for the filename. If the path is correct, the platforms are saved to the file.

The last selection from the main menu allows a user to return to the SELECT_AREA menu.

4. Operating Menus

a. Driving

There is only one menu that makes up the driving menu set. This menu is called OPERATE_DRIVE. This menu contains the seven selections as follows:

- DO NOTHING
- RETURN TO MAIN MENU
- CHANGE ALL PLATFORMS' SPEEDS
- EXIT THE PROGRAM
- ENTER 4SIGHT (RESIZE OPTIONS)
- POP WINDOWS
- ADVANCED OPTIONS

The first selection is provided in case the user pushes the right mouse button and he does not desire to do anything. The second selection allows the user to return to the main menu.

The third selection causes another menu to pop up that allows the user to select a speed for all the platforms currently in the simulation. The allowable speeds are from zero to 65 miles per hour. There is also a selection that will do nothing and return directly to the simulation. Changing all the speeds is convenient when the user wants to have a convoy of platforms proceed at identical speeds. Also, by selecting zero miles per hour, all platforms are effectively frozen and their configuration can be studied by viewing them from a FOGM missile or other platform.

The POP WINDOWS selection brings the four windows of the simulation into view if any of them are obscured from view by other processes that are running on the machine.

The ADVANCED OPTIONS selection brings up the following menu:

- TOGGLE SINGLE/DOUBLE BUFFER MODE
- TARGETING MODE TEST (ONCE)
- TERRAIN DRAWING OPTIONS

The first selection toggles the graphics hardware between singlebuffer and doublebuffer modes. In doublebuffer mode, all drawing is done in a separate area of memory from the display memory. When the function *swapbuffers*() is called, the pointer to this area and the pointer to the display buffer are switched, thereby swapping the new picture for the old picture. This is how smooth motion is simulated. If a user is interested in what order the individual picture elements are drawn on the screen, then by selecting singlebuffer mode, he can see the pictures while they are being drawn.

Targeting mode test allows a user to see how the simulation determines if a target is in the crosshairs of the FOGM missile during targeting. After selecting the option, the next time targeting is attempted, the view will be cleared to white and all

visible platforms will be drawn without lighting, shading, or hidden surface removal. The resulting picture is displayed for three seconds and then normal operation commences. This option is reset each time it is used.

The TERRAIN DRAWING OPTIONS option is a roll-off menu. When the user moves the cursor towards the right side of the words TERRAIN DRAWING OPTIONS, the following menu is displayed:

- DETAILED TERRAIN
- DISTANCE ATTENUATION - NORMAL
- DISTANCE ATTENUATION - BOUNDARIES DISPLAYED

The default terrain drawing option is DISTANCE ATTENUATION - NORMAL. This drawing option establishes three zones in front of the driven platform and reduces the number of polygons that are displayed in each zone. The zone closest to the viewer is displayed with 100x100 meter polygons, the greatest resolution available. The next zone uses 200x200 meter polygons and the last zone uses 400x400 meter polygons. The selection DISTANCE ATTENUATION - BOUNDARIES DISPLAYED draws the boundaries between zones in cyan so the user can see where they are. The selection for DETAILED TERRAIN draws 100x100 meter polygons throughout the three zones. Users notice a significant decrease in the frames per second rate when this option is selected. If singlebuffer mode is also enabled during detailed terrain drawing, the algorithm that is used to draw the terrain becomes more obvious.

b. Flying

There are three menus that make up the flying menu set. These menus are called OPERATE_FLY_ONE, OPERATE_FLY_TWO, and OPERATE_FLY _THREE. This menu contains the seven selections as follows:

- DO NOTHING
- DETACH/RESUME OPERATING
- RETURN TO MAIN MENU

- CHANGE ALL PLATFORMS' SPEEDS
- EXIT THE PROGRAM
- ENTER 4SIGHT (RESIZE OPTIONS)
- TOGGLE TARGET TRACKING
- ADVANCED OPTIONS

Many of these options are exact duplicates of the options on the driving menus. However, the DETACH/RESUME OPERATING and TOGGLE TARGET TRACKING options are different.

The DETACH/RESUME OPERATING option allows a user to detach the cursor from the simulation while flying. During flying, the cursor is restricted to the simulation window because the mouse controls where the nose camera of the FOGM missile is pointed. Using this option, the user can point the camera where he wants to look and then free the mouse. To return to the simulation, the user must select the same option once again.

If the user has a ground platform in the crosshairs of the FOGM missile and he wants to target it, he must make the TOGGLE TARGET TRACKING selection from the menu. If a platform was in the crosshairs, then the missile will lock on and track the platform. If the user wants to release the missile from tracking mode then another selection will turn off target tracking.

C. Dials[5]

The dial box that is supplied by SGI has eight dials numbered from zero to seven. They are organized in two columns and four rows. The numbering scheme is from left

---

[5]The code for initializing the dials is contained in the following files: setcontrols.c and setcontrols_fogm.c. Code for handling input from the dials' movements is contained in the following files: handlecontrols.c, handlecontrols_fogm.c, and handlecontrols_partial.c.

to right, bottom to top so the lower left dial is zero, the lower right is one and the upper right is seven.

The Moving Platform Simulator uses these dials in basically two configurations; one for driving and one for flying.

1. Driving Dial Configuration

The dials for driving are configured as follows:

- DIAL 0 - Course
- DIAL 1 - Viewing Direction
- DIAL 2 - Speed
- DIAL 3 - Tilt
- DIAL 4 - Hour of the Day
- DIAL 5 - Month of the Year
- DIAL 6 - Not Used
- DIAL 7 - Not Used

The course is the direction of travel of the platform which is displayed in degrees. The viewing direction is the direction the driver's head is looking left to right in relation to the course. When the course is changed, the viewing angle changes accordingly. Speed is the speed of the platform in miles per hour. Tilt is where the driver is looking up and down. The hour of the day and month of the year determine the location, color and intensity of the sun. Figure A.1 is a picture of the dial box with the dials labeled for driving.

2. Flying Dial Configuration

The dials for flying are configured as follows:

- DIAL 0 - Course
- DIAL 1 - Altitude
- DIAL 2 - Speed
- DIAL 3 - Not Used
- DIAL 4 - Hour of the Day
- DIAL 5 - Month of the Year

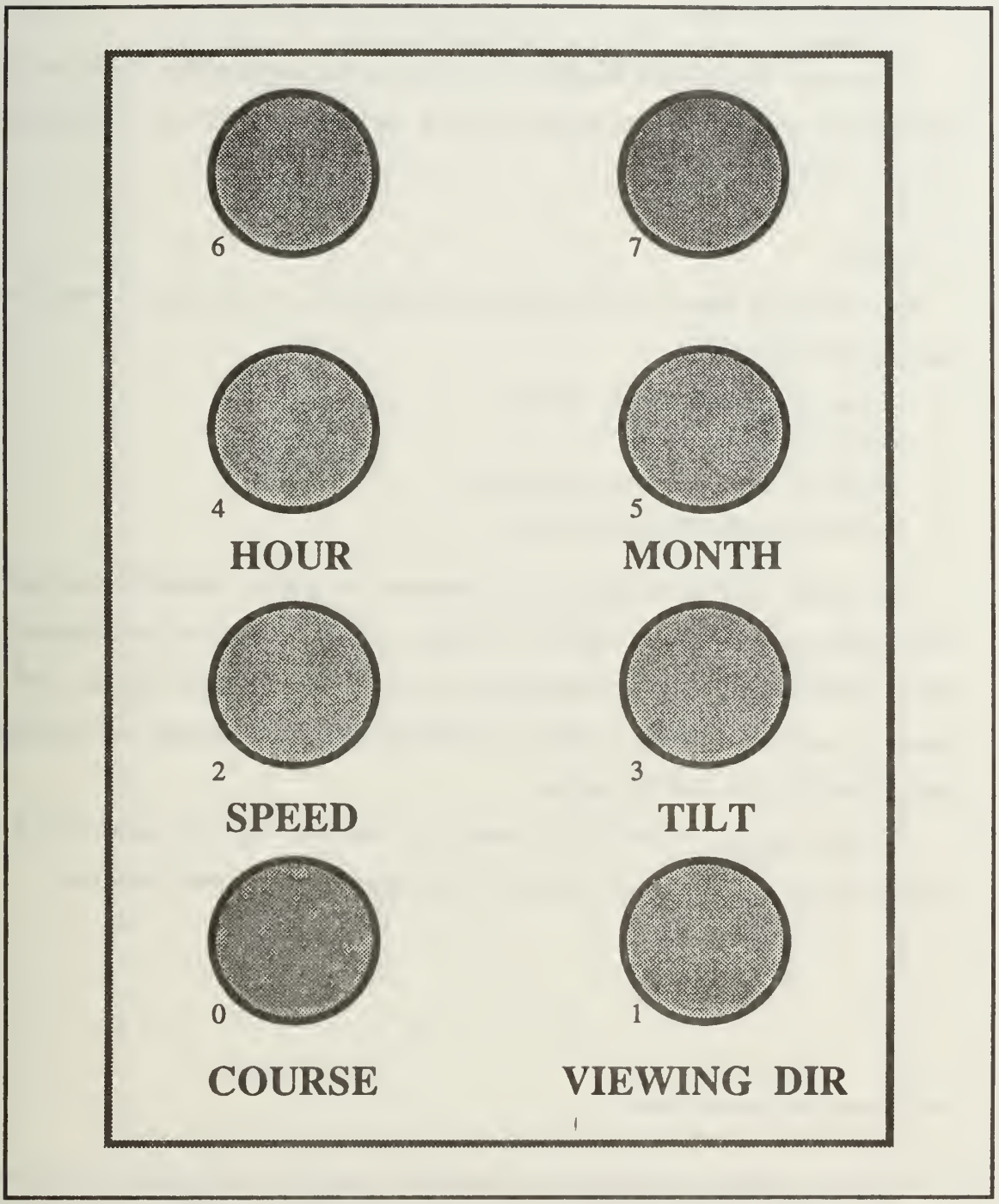

Figure A.1 Dial Box With Dials Labeled For Driving

- DIAL 6 - Not Used
- DIAL 7 - Not Used

Many of the dials are identical to the driving dial configuration except for altitude which is self-explanatory. Figure A.2 is a picture of the dial box with the dials labeled for flying.

D. Mouse[6]

The mouse has many uses throughout the simulation. Its use can be broken down into basically four groups:

- Popup menu activation and selection
- Operating area selection
- Platform icon placement and selection
- FOGM missile nose camera control

The mouse is used throughout the simulation to activate popup menus and to select options. One of these options is to select an area from the large database. A 10x10 kilometer red square is displayed on the 35x35 kilometer database and the mouse is used to move the square to the desired location. Platforms are placed and selected on the screen with the mouse.

The nose camera of the FOGM missile is controlled with the movement of the mouse. This gives the user very fine control over targeting and viewing direction.

---

[6]Code for handling the operations of the selections is contained in the file select_area_menu.c. Code for handling platform icon placement is contained in the files do_the_add.c and addveh.c. Code for handling FOGM missile nose camera control is contained in the files handlecontrols_fogm.c and handlecontrols_partial.c.

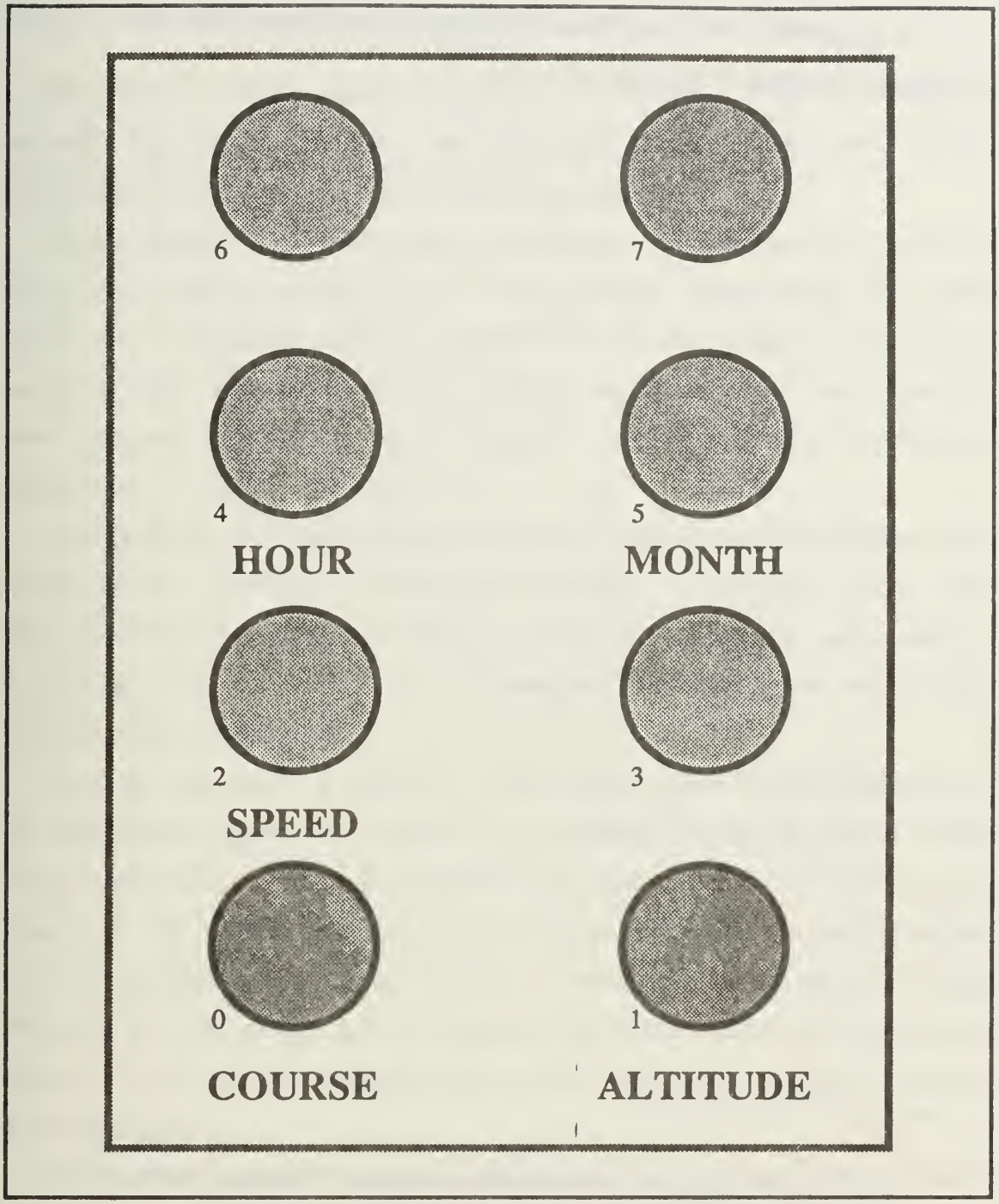

Figure A.2 Dial Box With Dials Labeled For Flying

E. Keyboard[7]

The keyboard is only used to accept filenames from the user. All other user input is through the popup menus, dials, or mouse.

[7]Code for handling filename input is contained in the files do_the_filename.c and do_char.c.

APPENDIX B MODULE/FILE ORGANIZATION

The top-level function *main*() is located in the file mps.c. Figure B.1 illustrates the major function calls that *main*() and *event*() make, and Figures B.2 and B.3 show the calls for *event_driving*() and *event_flying*() respectively.

Main() (Figure B.1) is responsible for initializing many different items including many data structures, cursors, colors, all light models, months, icons, and popup menus. The IRIS graphics hardware and windows are also initialized. The function *event*() is called after *main*() finishes initializing all these items. *Event*() does not return to *main*() until the user selects an option to exit the simulation at which time the function *exit_simulator*() is called.

Event() (Figure B.1) is the function that sets up everything in preparation for the actual real-time simulation. The introduction screens, area selection queries, two-dimensional map display, and all operations performed from the main menu (platform addition, deletion, and selection, etc.) are completed by *event*() before calling either *event_driving*() or *event_flying*().

Once the user selects a platform to operate, either *event_driving*() (Figure B.2) or *event_flying*() (Figure B.3) is called. All platforms except the FOGM missile cause *event_driving*() to be called. Each of these functions is the event loop of the simulation. The program continuously loops through the picture generating functions updating all platform positions and handling all user input from the mouse and dials. When the user selects RETURN TO MAIN MENU or the platform he is operating is destroyed, then control is returned to *event*(). The main menu is displayed and the loop begins again.

Figures B.1 through B.3 do not list all the functions that *main*(), *event*(), *event_driving*(), and *event_flying*() actually call. There are many other support

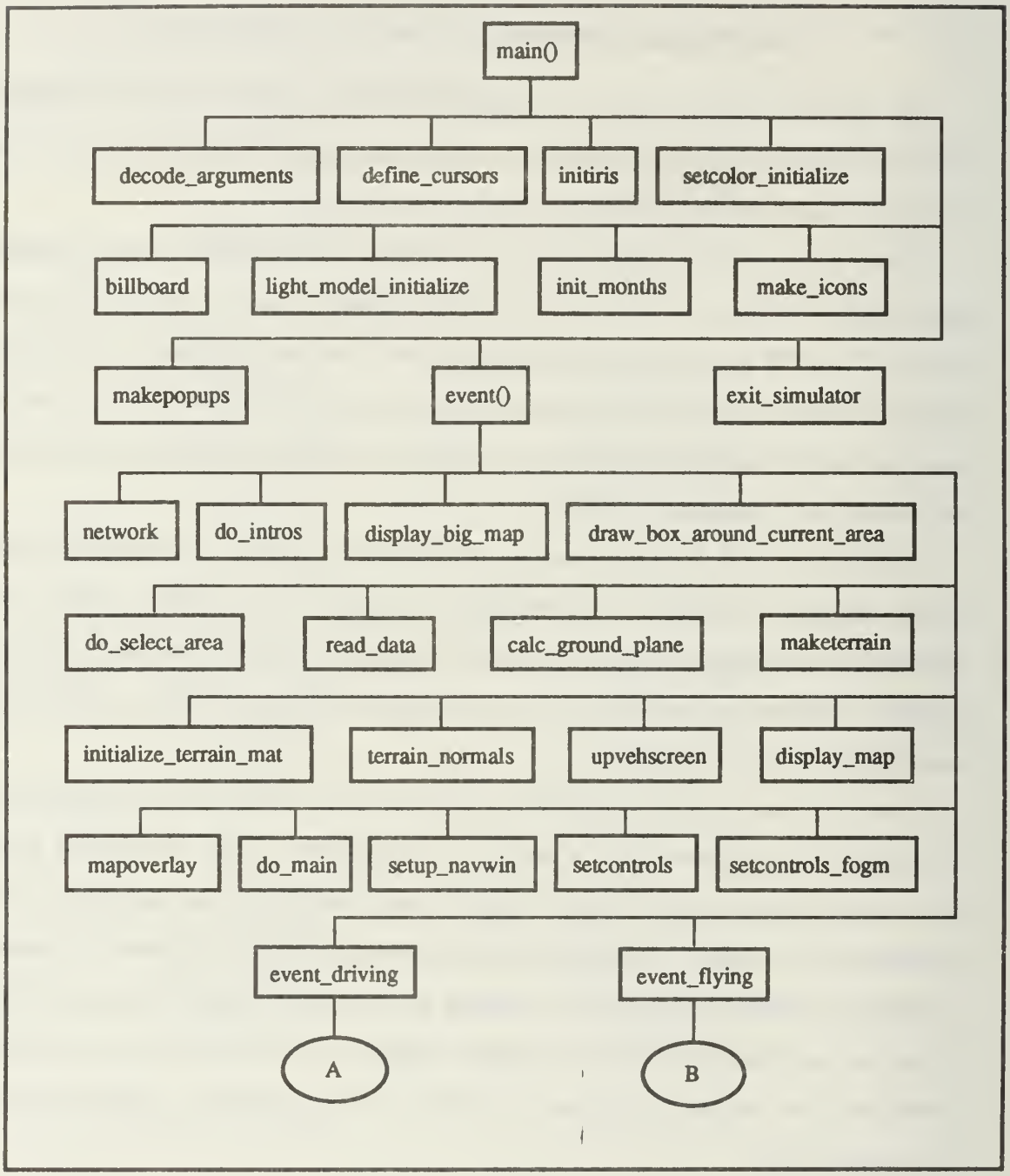

Figure B.1 Module Structure For main() and event()

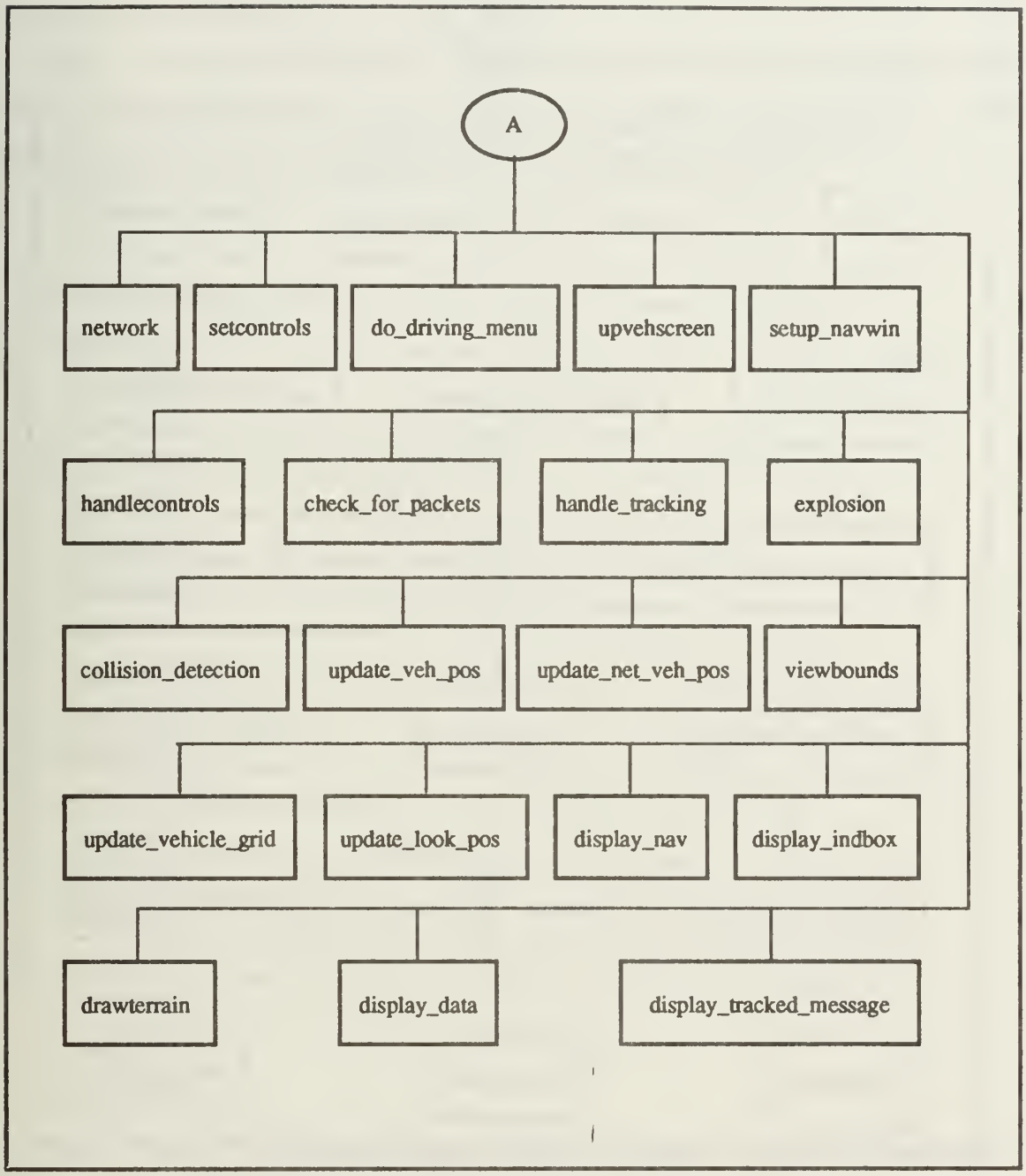

Figure B.2 Module Structure For event_driving()

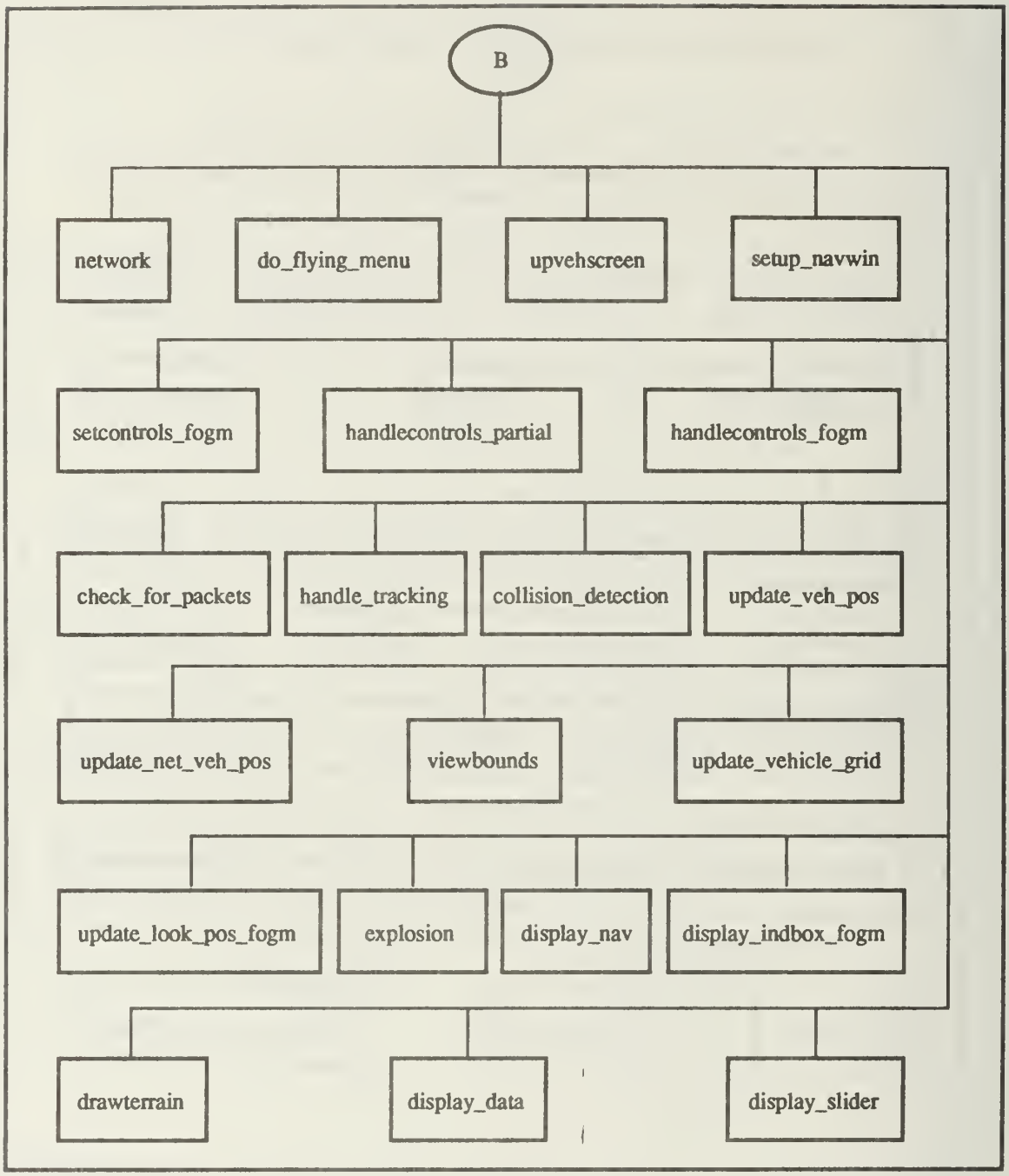

Figure B.3 Module Structure For event_flying()

routines that have not been listed. All the support routines and a short explanation of each are listed in Figure B.4.

Many of the files that have been discussed above require one or more header files. Many of these header files are system files, however there are some that are not. The following header files are specifically for the Moving Platform Simulator:

- color_scheme.h
- controls.h
- event_status.h
- files.h
- flamedata.h
- global.h
- intankdata.h
- jeepdata.h
- legend.h
- lightcons.h
- lightdefs.h
- missiledata.h
- mps.h
- network.h
- openjeepdata.h
- popups.h
- rollerdata.h
- tankdata.h
- terrain.h
- tiredata.h
- trackdata.h
- truckdata.h

Each of these files contains specific manifest constants and/or data structure declarations designed to support only those functions that require them. To determine which functions require a specific header file, refer to the makefile in Appendix C.

add_network_veh()	Adds a platform from a networked process.
addnode()	Creates vehgrid array list.
addnode_net()	Creates netvehgrid array list.
addveh()	Adds a local platform.
arcsine()	Returns arcsine of input parameter.
build_array()	Constructs vehgrid array list from linked list.
build_array_net()	Constructs netvehgrid array list from linked list.
calcwindows()	Calculates origins and sizes of windows.
center_cursor()	Centers the cursor in the designated window.
center_string_map()	Centers a string in the MAP window.
center_string_menu()	Centers a string in the MENU window.
clearwindow()	Clears a window to the given color.
compute_slope()	Computes the slope of a line.
compute_start_stop()	Computes information for drawterrain().
compute_sun_location()	Computes sun location based on month, hour.
compute_x_bounds()	Computes information for drawterrain().
compute_z_bounds()	Computes information for drawterrain().
computeavgfps()	Computes the average frames per second.
convert_to_dec_hr()	Converts to decimal hour.
convert_to_hr_min()	Converts to hours and minutes.
delete_veh()	Deletes a platform from the linked list.
display_intro_screen()	Displays an intro screen in the MAP window.
display_legend_for_big_map()	Displays the legend in the IND window.
display_legend_for_navbox()	Displays the legend in the NAV window.
do_capture()	Handles capturing platforms to a data file.
do_char()	Displays a character in the filename window.
do_change_speed()	Allows a user to change all platform speeds.
do_main_1()	Support routine for do_main().
do_main_2()	Support routine for do_main().
do_main_3()	Support routine for do_main().
do_main_4()	Support routine for do_main().
do_main_reset()	Clears all windows to black and draws 2D map.
do_resize()	Handles resizing selection.
do_quitting()	Handles quitting selection.
do_the_add()	Handles adding a platform.

Figure B.4 Support Functions

do_the_defaults()	Handles adding a default set of platforms.
do_the_delete()	Handles deleting one or all platforms.
do_the_filename()	Queries the user for a filename.
do_the_select()	Handles selecting a platform to operate.
drawflame()	Draws the flame for wrecks (obstacles).
drawgridbox()	Draws a box in the mapwin.
drawintank()	Draws the tank when you are operating a tank.
drawjeep()	Draws a covered jeep.
drawmissile()	Draws a missile.
drawopenjeep()	Draws an open jeep.
drawroller()	Draws the tank rollers.
drawtank()	Draws a tank.
drawtire()	Draws a tire.
drawtrack()	Draws a tank track.
drawtruck()	Draws a truck.
drawwreck()	Draws a wreck (obstacle).
error_handler()	Centralized routine to handle errors.
flamenormals()	Computes normals for the flame.
get_curr_fps()	Gets the current frames per second.
get_mouse_xy()	Finds the screen location of the mouse cursor.
gnd_level()	Calculates the elevation for a platform.
gridwindows()	Computes windowsx and windowsy (variables).
highlitegrid()	Highlights 1x1 km grids with platforms in them.
initveh()	Adds platform that is sent in the parameter list.
intanknormals()	Computes the normals for the intank.
jeepnormals()	Computes the normals for a covered jeep.
letter()	Draws a letter in the billboard.
lightdefs()	Defines materials, lights, and lighting models.
limit_cursor_pick()	Limits cursor for targeting attempt by FOGM.
limit_value()	Limits a value between lower and upper bound.
loadunit()	Loads a unit matrix onto the system stack.
missilenormals()	Computes normals for FOGM missile.
mousescreentoterrain()	Converts mouse screen coords to terrain coords.
mousescreentoworld()	Converts mouse screen coords to world coords.

Figure B.4 Support Functions - Continued

mouseterraintoscreen()	Converts mouse terrain coords to screen coords.
mouseworldtoscreen()	Converts mouse world coords to screen coords.
normalorient()	Computes normal and reorganizes vertices.
npoly_orient()	Orients polygons for backface polygon removal.
openjeepnormals()	Computes normals for an open jeep.
placewindow_sizes()	Sets aspect, min, and max for billboard window.
placewindows()	Calculates and opens all windows.
popwindow()	Pops a window into full view.
positionwindows()	Positions windows for winconstraints().
reset_tiltf()	Resets tiltf angle after releasing track mode.
ring_the_bell()	Rings the bell if not in silent mode.
rollernormals()	Computes normals for tank rollers.
select_grid_square()	Handles selection of 1x1 kilometer grid square.
semaphore()	Contains semaphore operations for networking.
set_popup_color()	Sets the popup menu color.
set_queue()	Queues input devices.
set_unqueue()	Unqueues input devices.
setcolor()	Sets RGBcolor().
setcolor_initialize()	Initializes the color information data structure.
setcursorcolor()	Sets the cursor color.
setwindow()	Does a winset() to the desired window.
setworldcoord()	Saves world coord for current window.
sincos()	Fast approximation for sine and cosine.
slowturn()	Causes a platform to turn slowly to the left.
switch_veh()	Returns a pointer to the selected platform.
tanknormals()	Computes normals for a tank.
tirenormals()	Computes normals for a tire.
tracknormals()	Computes normals for tank tracks.
trucknormals()	Computes normals for a truck.
tot_num_ground_veh()	Returns total number of ground platforms.
tot_num_veh()	Returns total number of platforms (all types).
vecdotp()	Vector dot product.
vecmag()	Returns magnitude of a vector.
zoomin()	Displays a 1x1 kilometer grid square.

Figure B.4 Support Functions - Continued

APPENDIX C MAKEFILE FOR THE MOVING PLATFORM SIMULATOR

```
CFLAGS     = -Zg -lm -O
CFLAGSLINK = -Zg -lm -lbsd -O
CFLAGSNET  = -Zg -lm -lbsd -O

MPSOBJS1 = addnode.o \
    addnode_net.o \
    add_network_veh.o \
    addveh.o \
    arcsine.o \
    billboard.o \
    build_array.o \
    build_array_net.o \
    calc_ground_plane.o \
    calcwindows.o \
    center_cursor.o \
    center_string_map.o \
    center_string_menu.o \
    check_for_packets.o \
    clearwindow.o \
    collision_detection.o \
    compute_slope.o \
    compute_start_stop.o \
    compute_sun_location.o \
    compute_x_bounds.o \
    compute_z_bounds.o \
    computeavgfps.o \
    convert_to_dec_hr.o \
    convert_to_hr_min.o \
    decode_arguments.o \
    define_cursors.o \
    delete_veh.o \
    display_big_map.o \
    display_data.o \
    display_indbox.o \
    display_indbox_fogm.o \
```

```
    display_intro_screen.o \
    display_legend_for_big_map.o \
    display_legend_for_navbox.o \
    display_map.o \
    display_nav.o \
    display_slider.o \
    display_tracked_message.o \
    do_capture.o \
    do_change_speed.o \
    do_char.o \
    do_driving_menu.o \
    do_flying_menu.o \
    do_intros.o \
    do_main.o \
    do_main_1.o \
    do_main_2.o \
    do_main_3.o \
    do_main_4.o \
    do_main_reset.o \
    do_quitting.o \
    do_resize.o \
    do_select_area.o \
    do_the_add.o \
    do_the_defaults.o \
    do_the_delete.o \
    do_the_filename.o \
    do_the_select.o
MPSOBJS2 = draw_box_around_current_area.o \
    drawflame.o \
    drawgridbox.o \
    drawintank.o \
    drawjeep.o \
    drawmissile.o \
    drawopenjeep.o \
    drawroller.o \
    drawtank.o \
    drawterrain.o \
    drawtire.o \
```

drawtrack.o \
drawtruck.o \
drawwreck.o \
error_handler.o \
event.o \
event_driving.o \
event_flying.o \
exit_simulator.o \
explosion.o \
flamenormals.o \
get_curr_fps.o \
get_mouse_xy.o \
gnd_level.o \
gridwindows.o \
handle_tracking.o \
handlecontrols.o \
handlecontrols_fogm.o \
handlecontrols_partial.o \
highlitegrid.o \
init_months.o \
initialize_terrain_mat.o \
initiris.o \
initveh.o \
intanknormals.o \
jeepnormals.o \
letter.o \
light_model_initialize.o \
lightdefs.o \
limit_cursor_pick.o \
limit_value.o \
loadunit.o \
makeicons.o \
makepopups.o \
maketerrain.o \
mapoverlay.o \
missilenormals.o \
mousescreentoterrain.o \
mousescreentoworld.o \

```
    mouseterraintoscreen.o \
    mouseworldtoscreen.o \
    mps.o
MPSOBJS3 = network.o \
    normalorient.o \
    npoly_orient.o \
    openjeepnormals.o \
    placewindows.o \
    placewindow_sizes.o \
    popwindow.o \
    positionwindows.o \
    read_data.o \
    reset_tiltf.o \
    ring_the_bell.o \
    rollernormals.o \
    select_an_area.o \
    select_grid_square.o \
    semaphore.o \
    set_popup_color.o \
    set_queue.o \
    set_unqueue.o \
    setcolor_initialize.o \
    setcolor.o \
    setcontrols.o \
    setcontrols_fogm.o \
    setcursorcolor.o \
    setup_navwin.o \
    setwindow.o \
    setworldcoord.o \
    sincos.o \
    slowturn.o \
    switch_veh.o \
    tanknormals.o \
    terrainnormals.o \
    tirenormals.o \
    tracknormals.o \
    trucknormals.o \
    tot_num_ground_veh.o \
```

```
        tot_num_veh.o \
        tracking_check.o \
        update_look_pos.o \
        update_look_pos_fogm.o \
        update_net_veh_pos.o \
        update_veh_pos.o \
        update_vehicle_grid.o \
        upvehscreen.o \
        vecdotp.o \
        vecmag.o \
        viewbounds.o \
        zoomin.o
NETWORK_RECEIVEOBJS = semaphore.o \
                network_receive.o
MPSHDRS = addnode.o \
        addnode_net.o \
        add_network_veh.o \
        addveh.o \
        arcsine.o \
        billboard.o \
        build_array.o \
        build_array_net.o \
        calcwindows.o \
        center_string_map.o \
        center_string_menu.o \
        check_for_packets.o \
        collision_detection.o \
        compute_start_stop.o \
        compute_sun_location.o \
        compute_x_bounds.o \
        compute_z_bounds.o \
        define_cursors.o \
        delete_veh.o \
        display_big_map.o \
        display_indbox.o \
        display_indbox_fogm.o \
        display_intro_screen.o \
        display_legend_for_big_map.o \
```

```
display_legend_for_navbox.o \
display_map.o \
display_nav.o \
display_slider.o \
display_tracked_message.o \
do_capture.o \
do_change_speed.o \
do_char.o \
do_driving_menu.o \
do_flying_menu.o \
do_intros.o \
do_main.o \
do_main_1.o \
do_main_2.o \
do_main_3.o \
do_main_4.o \
do_main_reset.o \
do_quitting.o \
do_resize.o \
do_select_area.o \
do_the_add.o \
do_the_defaults.o \
do_the_delete.o \
do_the_filename.o \
do_the_select.o \
draw_box_around_current_area.o \
drawgridbox.o \
drawterrain.o \
event.o \
event_driving.o \
event_flying.o \
exit_simulator.o \
explosion.o \
get_curr_fps.o \
get_mouse_xy.o \
gnd_level.o \
gridwindows.o \
handle_tracking.o \
```

handlecontrols.o \
handlecontrols_fogm.o \
handlecontrols_partial.o \
highlitegrid.o \
initiris.o \
initveh.o \
intank.o \
limit_cursor_pick.o \
makeicons.o \
makepopups.o \
maketerrain.o \
mapoverlay.o \
mousescreentoterrain.o \
mousescreentoworld.o \
mouseterraintoscreen.o \
mps.o \
network.o \
placewindows.o \
placewindow_sizes.o \
popwindow.o \
positionwindows.o \
read_data.o \
reset_tiltf.o \
select_an_area.o \
select_grid_square.o \
set_popup_color.o \
set_queue.o \
setcontrols.o \
setcontrols_fogm.o \
setup_navwin.o \
sincos.o \
slowturn.o \
switch_veh.o \
terrainnormals.o \
tracking_check.o \
tot_num_ground_veh.o \
tot_num_veh.o \
update_look_pos.o \

```
        update_look_pos_fogm.o \
        update_net_veh_pos.o \
        update_veh_pos.o \
        update_vehicle_grid.o \
        upvehscreen.o \
        vecdotp.o \
        viewbounds.o \
        zoomin.o
POPUPHDRS = compute_start_stop.o \
        do_change_speed.o \
        do_driving_menu.o \
        do_flying_menu.o \
        do_intros.o \
        do_main.o \
        do_main_1.o \
        do_main_2.o \
        do_main_3.o \
        do_main_4.o \
        do_quitting.o \
        do_the_add.o \
        do_the_defaults.o \
        do_the_delete.o \
        do_the_select.o \
        drawterrain.o \
        event.o \
        event_flying.o \
        makepopups.o \
        mps.o \
        set_popup_color.o \
        set_queue.o
COLORSCHEMEHDRS = display_indbox.o \
                display_indbox_fogm.o \
                display_intro_screen.o \
                do_select_area.o \
                drawterrain.o \
                event.o \
                mps.o \
                network.o \
```

```
            setcolor_initialize.o
FILEHDRS = display_big_map.o \
        do_the_defaults.o \
        maketerrain.o \
        read_data.o
EVENTSTATUSHDRS = collision_detection.o \
            do_driving_menu.o \
            do_flying_menu.o \
            do_main.o \
            do_select_area.o \
            do_the_select.o \
            event.o \
            event_driving.o \
            event_flying.o \
            handle_tracking.o \
            set_queue.o
CONTROLSHDRS = do_change_speed.o \
            drawterrain.o \
            event.o \
            handlecontrols.o \
            handlecontrols_fogm.o \
            handlecontrols_partial.o \
            setcontrols.o \
            setcontrols_fogm.o \
            update_look_pos.o
LEGEND     = display_legend_for_big_map.o \
            display_legend_for_navbox.o
NETWORKHDRS = check_for_packets.o \
            collision_detection.o \
            do_change_speed.o \
            do_flying_menu.o \
            do_intros.o \
            do_main.o \
            event.o \
            event_driving.o \
            event_flying.o \
            network.o \
            network_receive.o \
```

```
        tracking_check.o
LIGHTCONS = compute_sun_location.o \
        drawintank.o \
        drawflame.o \
        drawjeep.o \
        drawmissile.o \
        drawopenjeep.o \
        drawroller.o \
        drawtank.o \
        drawterrain.o \
        drawtire.o \
        drawtrack.o \
        drawtruck.o \
        drawwreck.o \
        initialize_terrain_mat.o \
        lightdefs.o \
        mps.o
LIGHTDEFS = compute_sun_location.o \
        lightdefs.o
INTANKDATA = mps.o
FLAMEDATA = mps.o
JEEPDATA = mps.o
OPENJEEPDATA = mps.o
MISSILEDATA= mps.o
ROLLERDATA = mps.o
TANKDATA = mps.o
TIREDATA = mps.o
TERRAIN = compute_start_stop.o \
        drawterrain.o
TRACKDATA = mps.o
TRUCKDATA = mps.o

all: mps network_receive

network.o: network.c
   cc -c network.c $(CFLAGSNET)
mps_module_1.o : $(MPSOBJS1)
   ld -r $(MPSOBJS1) -o mps_module_1.o
```

```
mps_module_2.o : $(MPSOBJS2)
    ld -r $(MPSOBJS2) -o mps_module_2.o

mps_module_3.o : $(MPSOBJS3)
    ld -r $(MPSOBJS3) -o mps_module_3.o

mps: mps_module_1.o mps_module_2.o mps_module_3.o
  cc -o mps mps_module_1.o mps_module_2.o mps_module_3.o $(CFLAGSLINK)

network_receive: $(NETWORK_RECEIVEOBJS)
    cc -o network_receive $(NETWORK_RECEIVEOBJS) $(CFLAGSNET)

mps.o: global.h
$(MPSHDRS): mps.h
$(POPUPHDRS): popups.h
$(COLORSCHEMEHDRS): color_scheme.h
$(FILEHDRS): files.h
$(EVENTSTATUSHDRS): event_status.h
$(CONTROLSHDRS): controls.h
$(LEGEND) : legend.h
$(NETWORKHDRS): network.h
$(LIGHTCONS)  : lightcons.h
$(LIGHTDEFS)  : lightdefs.h
$(INTANKDATA) : intankdata.h
$(FLAMEDATA)  : flamedata.h
$(JEEPDATA)   : jeepdata.h
$(OPENJEEPDATA)   : openjeepdata.h
$(MISSILEDATA): missiledata.h
$(ROLLERDATA) : rollerdata.h
$(TANKDATA)   : tankdata.h
$(TERRAINDATA)   : terraindata.h
$(TIREKDATA)   : tiredata.h
$(TRACKDATA)   : trackdata.h
$(TRUCKDATA)   : truckdata.h
```

APPENDIX D PROCEDURE FOR ADDING ADDITIONAL PLATFORMS TO MPS

Step 1: Using graph paper or any other means, locate all vertices of all polygons that are needed to create the new platform. Use any world coordinate system that is the most logical for the new platform. This is perhaps the hardest part of designing a new platform.

Step 2: Using the development tool **obj**[8], insure that all polygons interconnected correctly and create all polygon normals insuring that they are correct.

Step 3: Three files need to be created for the new platform. In this example we will assume that we are designing a cobra helicopter. The three files that need to be created are drawcobra.c, cobradata.h, and cobranormals.c.

Drawcobra.c contains the commands that actually draw the polygons. Each polygon in the cobra helicopter needs commands similar to those in Figure D.1. These

```
/* Right front side panel */
n3f(pncobra39);
bgnpolygon();
        v3f(pcobra39[0]);
        v3f(pcobra39[1]);
        v3f(pcobra39[2]);
        v3f(pcobra39[3]);
endpolygon();
```

Figure D.1 Example Commands to Draw a Single Polygon For a Platform

sequence of commands draws ONE of the many polygons that make up the cobra. The

[8]**Obj** is a tool that allows a collection of one or more polygons to be viewed from anywhere in three dimensional space. It was instrumental in designing and updating platforms for lighting. **Obj** was written by Mr. David Jennings at the same time he and CPT Mark Fichten were working on the Moving Platform Simulator.

n3f() functions tells the system where the polygon's normal is so that lighting can be applied correctly. The *bgnpolygon*() and *endpolygon*() surround *v3f*() function calls that define where the polygon's vertices are. The number 39 in the example simply means that this is the 39^{th} polygon of the cobra and since four vertices were declared, the polygon has four sides.

Cobradata.h is a header file that contains the declarations of the variables such as pncobra39 from the above example. All these variables are initialized with the actual points in space of the polygon in question. These points in space come from step one above.

Cobranormals.c contains calls to the function *normalorient*() which returns a normal vector for the polygon in the direction of the light source. An interior point of the polygon must also be passed into the function as a parameter.

After these three files are created, they are ready to be incorporated into the Moving Platform Simulator. Up to this point, no changes have been made to the actual simulator itself.

Step 4: Add the variable declarations for the cobra normals to the header file global.h.

Step 5: Add the statement #include "cobradata.h" to the top of the file mps.c. This gives the variable declarations in cobradata.h global visibility.

Step 6: Change the makefile to reflect the new platform. Copy and modify the definitions for an existing platform.

Step 7: Add all the appropriate manifest constants to the header file mps.h. Use an existing platform to model the changes.

Step 8: Set up the lighting characteristics for the new platform by updating the header files lightdefs.h and lightcons.h.

Step 9: Issue the command **fgrep TANK *.c** in the directory that MPS resides. Changes must be made to all files that contain references to the **TANK**. Most of

these changes will require an addition to a **switch** statement. Some of the files that will require changes include:

- build_array.c
- build_array_net.c
- define_cursors.c
- display_slider.c
- do_the_add.c
- do_the_defaults.c
- do_the_select.c
- event.c
- lightdefs.c
- makeicons.c
- makepopups.c
- mps.c
- tot_num_ground_veh.c
- tot_num_veh.c
- tracking_check.c
- update_net_veh_pos.c
- update_veh_pos.c

Step 10: This completes the procedure to add a platform to the Moving Platform Simulator. Good luck!

APPENDIX E TERRAIN DISPLAY DETAILS

Each grid square that is displayed in the Moving Platform Simulator is drawn as two triangles. Each triangle is labeled with either an **L** for lower or **U** for upper, thus denoting its location within the grid square. Also the vertices of each triangle are numbered from zero to two in a counter-clockwise order. This is shown for each grid square in the near group of Figure E.1.

Each grid square of the near group is displayed when the terrain is drawn. First the coordinates of the lower triangular vertices are drawn, followed by the coordinates of the upper triangle. No triangular filling is needed between rows or columns of grid squares since every square is displayed, and none are grouped to form larger squares.

A. NORTH DISPLAY

When the viewer is looking north, the terrain is displayed moving from minimum to maximum z, displaying rows of grid squares from minimum to maximum x. Figure E.1 shows a simplified view of how the grid squares are combined in each drawing group. Four 100 x 100 meter grid squares are combined to form each large grid square in the mid group. Each large grid square in the far group contains 16 100 x 100 meter grid squares.

The appropriate coordinates of a vertex from the small grid square are sent to the *v3f()* command in order to display the large triangles. For example if the current grid square (x,z) is #1 in Figure E.1, the following coordinates are sent to display the large lower triangle:

- (x,z) lower triangle vertex 0
- (x+1,z) lower triangle vertex 1
- (x,z+1) lower triangle vertex 2

The upper triangle is displayed after the lower triangle is drawn. The following coordinates are used:

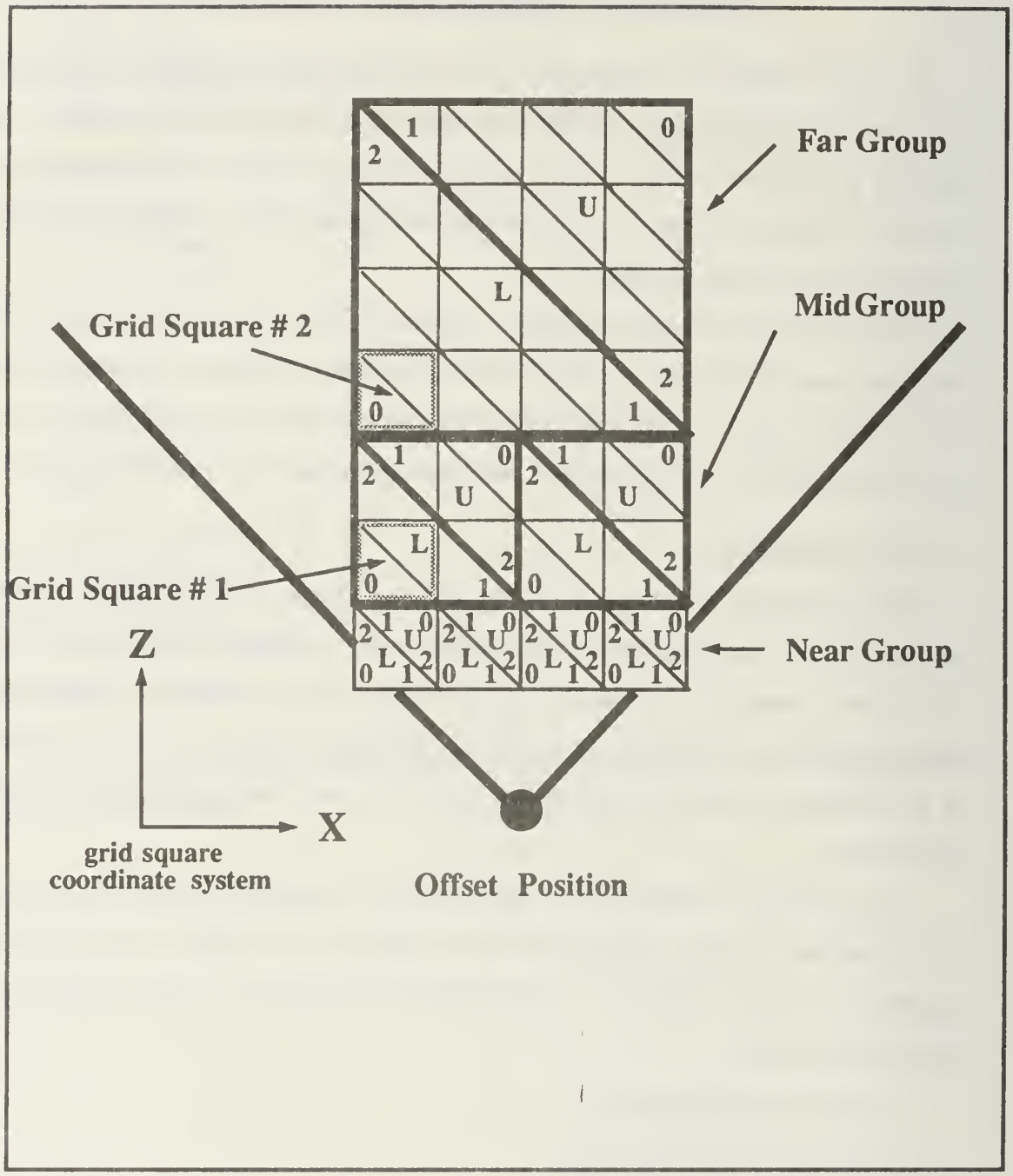

Figure E.1 Grid Square Display Looking North

- (x+1,z+1) upper triangle vertex 0
- (x,z+1) upper triangle vertex 1
- (x+1,z) upper triangle vertex 2

Table E.1 shows the vertex coordinates needed to display each triangle of the mid and far groups. This table assumes the current grid square is #1 for the mid group and #2 for the far group.

When grid squares are grouped together to form larger ones, **holes** may appear between the groups. These **holes** are filled by drawing triangular regions where the groups meet. Again using grid square #1 in the mid group and grid square #2 in the far group, Table E.2 shows what vertices are needed to fill the gaps between groups.

B. SOUTH, EAST, AND WEST DISPLAYS

The procedure used for the north display is also used to display terrain while viewing the other directions. Only the order and vertex numbers change. Figure E.2 shows the display groups while looking south, and Tables E.3 and E.4 give the appropriate vertices needed.

Figure E.3, Tables E.5 and E.6 summarize the east display. Figure E.4, Tables E.7 and E.8 show the vertices to display the terrain while looking west.

TABLE E.1 VERTEX COORDINATES LOOKING NORTH

CURRENT GRID SQUARE # 1

LOWER VERTEX 0	LOWER VERTEX 1	LOWER VERTEX 2	UPPER VERTEX 0	UPPER VERTEX 1	UPPER VERTEX 2
x z	x1 z	x z1	x1 z1	x z1	x1 z

CURRENT GRID SQUARE # 2

LOWER VERTEX 0	LOWER VERTEX 1	LOWER VERTEX 2	UPPER VERTEX 0	UPPER VERTEX 1	UPPER VERTEX 2
x z	x3 z	x z3	x3 z3	x z3	x3 z

x1 = x + 1
x3 = x + 3
z1 = z + 1
z3 = z + 3

TABLE E.2 VERTEX COORDINATES FOR FILLING HOLES LOOKING NORTH

CURRENT GRID SQUARE # 1

VERTEX 0	VERTEX 1	VERTEX 2
x z L vertex 0	x z L vertex 1	x1 z L vertex 1

CURRENT GRID SQUARE # 2

VERTEX 0	VERTEX 1	VERTEX 2
x z L vertex 0	x1 z L vertex 1	x3 z L vertex 1

x1 = x + 1

x3 = x + 3

L = lower triangle

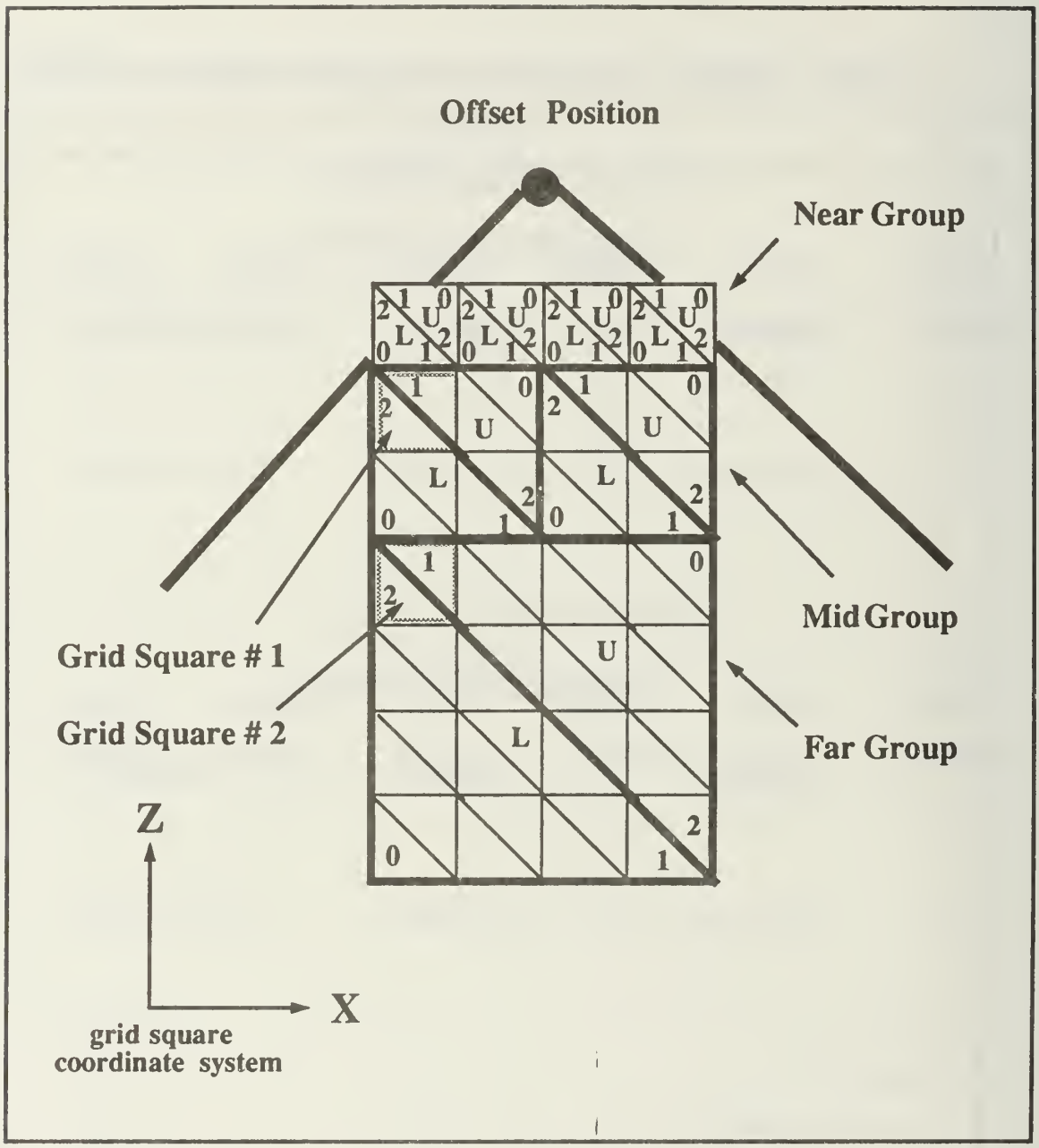

Figure E.2 Grid Square Display Looking South

TABLE E.3 VERTEX COORDINATES LOOKING SOUTH

CURRENT GRID SQUARE # 1

LOWER VERTEX 0	LOWER VERTEX 1	LOWER VERTEX 2	UPPER VERTEX 0	UPPER VERTEX 1	UPPER VERTEX 2
x zm1	x1 zm1	x z	x1 z	x z	x1 zm1

CURRENT GRID SQUARE # 2

LOWER VERTEX 0	LOWER VERTEX 1	LOWER VERTEX 2	UPPER VERTEX 0	UPPER VERTEX 1	UPPER VERTEX 2
x zm3	x3 zm3	x z	x3 z	x z	x3 zm3

x1 = x + 1
x3 = x + 3
zm1 = z - 1
zm3 = z - 3

TABLE E.4 VERTEX COORDINATES FOR FILLING HOLES LOOKING SOUTH

CURRENT GRID SQUARE # 1

VERTEX 0	VERTEX 1	VERTEX 2
x z U vertex 1	x z U vertex 0	x1 z U vertex 0

CURRENT GRID SQUARE # 2

VERTEX 0	VERTEX 1	VERTEX 2
x z U vertex 1	x1 z U vertex 0	x3 z U vertex 0

x1 = x + 1

x3 = x + 3

U = upper triangle

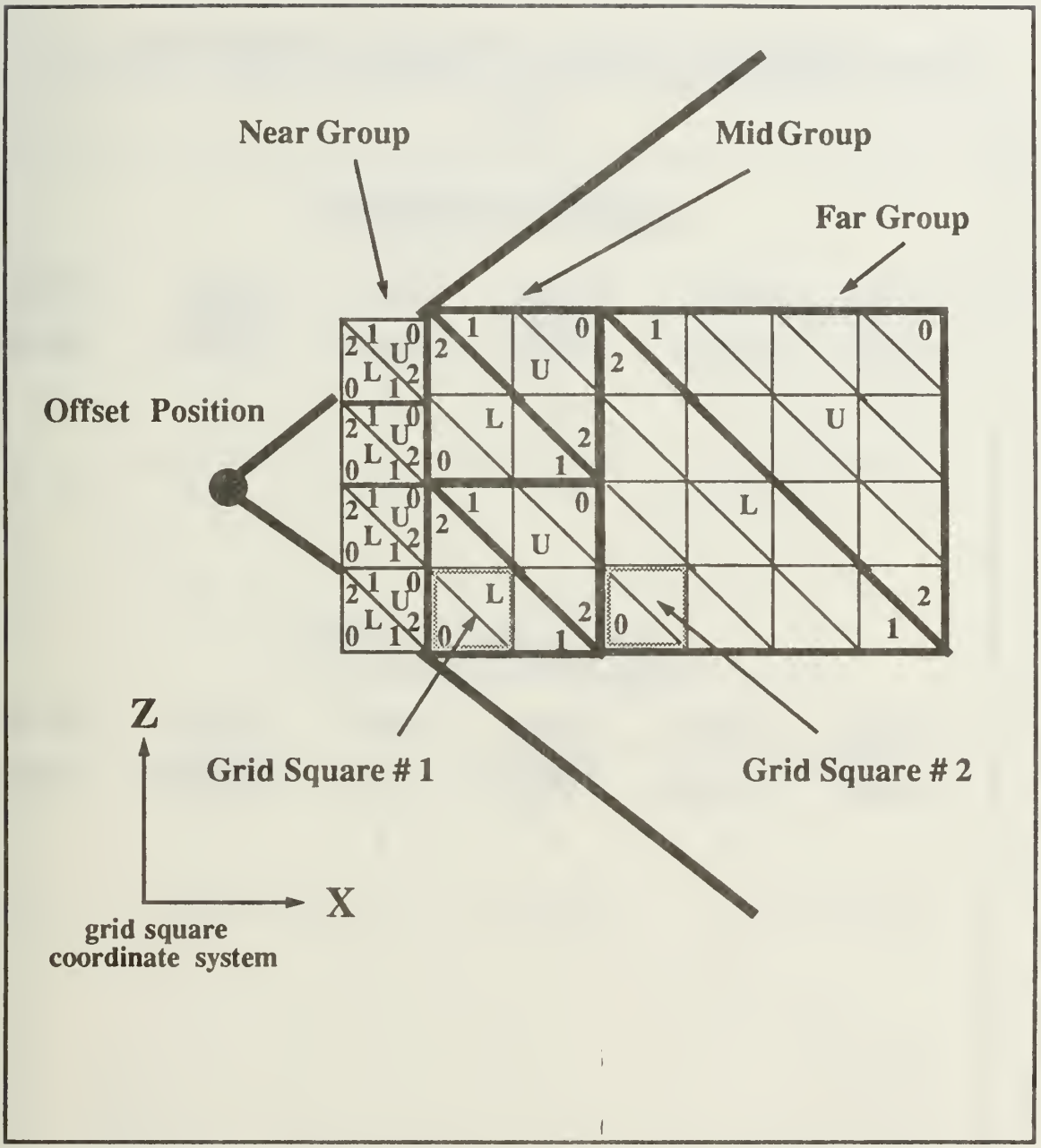

Figure E.3 Grid Square Display Looking East

TABLE E.5 VERTEX COORDINATES LOOKING EAST

CURRENT GRID SQUARE # 1

LOWER VERTEX 0	LOWER VERTEX 1	LOWER VERTEX 2	UPPER VERTEX 0	UPPER VERTEX 1	UPPER VERTEX 2
x z	x1 z	x z1	x1 z1	x z1	x1 z

CURRENT GRID SQUARE # 2

LOWER VERTEX 0	LOWER VERTEX 1	LOWER VERTEX 2	UPPER VERTEX 0	UPPER VERTEX 1	UPPER VERTEX 2
x z	x3 z	x z3	x3 z3	x z3	x3 z

x1 = x + 1
z1 = z + 1
x3 = x + 3
z3 = z + 3

TABLE E.6 VERTEX COORDINATES FOR FILLING HOLES LOOKING EAST

CURRENT GRID SQUARE # 1

VERTEX 0	VERTEX 1	VERTEX 2
x z L vertex 0	x z L vertex 2	x z1 L vertex 2

CURRENT GRID SQUARE # 2

VERTEX 0	VERTEX 1	VERTEX 2
x z L vertex 0	x z1 L vertex 2	x z3 L vertex 2

z1 = z + 1

z3 = z + 3

L = lower triangle

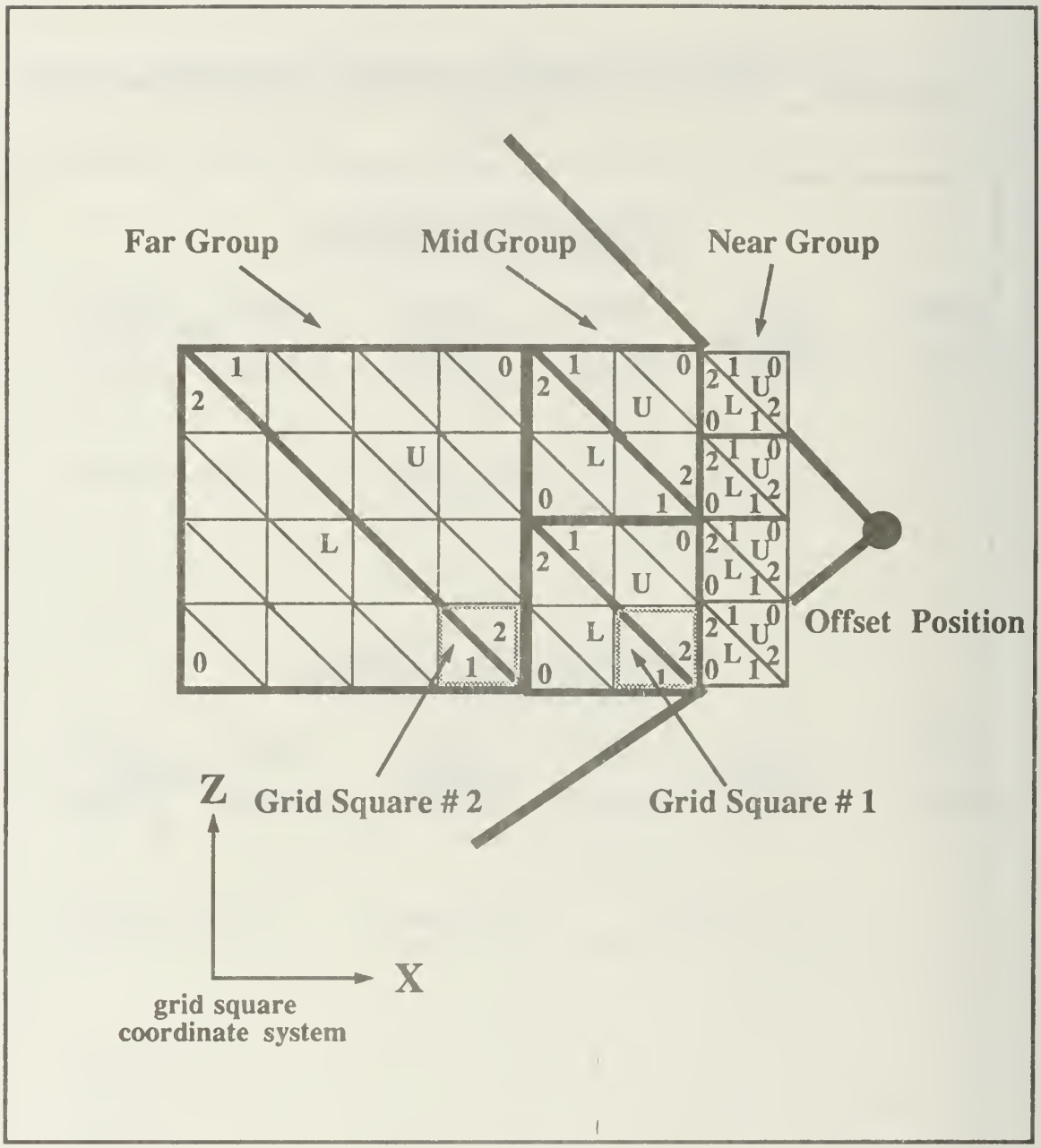

Figure E.4 Grid Square Display Looking West

TABLE E.7 VERTEX COORDINATES LOOKING WEST

CURRENT GRID SQUARE # 1

LOWER VERTEX 0	LOWER VERTEX 1	LOWER VERTEX 2	UPPER VERTEX 0	UPPER VERTEX 1	UPPER VERTEX 2
xm1 z	x z	xm1 z1	x z1	xm1 z1	x z

CURRENT GRID SQUARE # 2

LOWER VERTEX 0	LOWER VERTEX 1	LOWER VERTEX 2	UPPER VERTEX 0	UPPER VERTEX 1	UPPER VERTEX 2
xm3 z	x z	xm3 z3	x z3	xm3 z3	x z

xm1 = x - 1
xm3 = x - 3
z1 = z + 1
z3 = z + 3

TABLE E.8 VERTEX COORDINATES FOR FILLING HOLES LOOKING WEST

CURRENT GRID SQUARE # 1

VERTEX 0	VERTEX 1	VERTEX 2
x z U vertex 2	x z U vertex 0	x z1 U vertex 0

CURRENT GRID SQUARE # 2

VERTEX 0	VERTEX 1	VERTEX 2
x z U vertex 2	x z1 U vertex 0	x z3 U vertex 0

z1 = z + 1

z3 = z + 3

U = upper triangle

LIST OF REFERENCES

1. Blau, Ricki, and others, Panel session on "What Can We Learn by Benchmarking Graphics Systems?," *Computer Graphics*, SIGGRAPH 1988 Conference Proceedings, v. 22, no. 4, August 1988.
2. Akeley, Kurt and Jermoluk, Tom, "High-Performance Polygon Rendering," *Computer Graphics*, SIGGRAPH 1988 Conference Proceedings, v. 22, no. 4, August 1988.
3. Ardent Computer Corporation, *Ardent Computer Corporation Titan Specifications*, San Jose, California, 1988.
4. Apgar, Brian, Bersack, Bret and Mammen, Abraham, "A Display System for the Stellar™ Graphics Supercomputer Model GS1000™," *Computer Graphics*, SIGGRAPH 1988 Conference Proceedings, v. 22, no. 4, August 1988.
5. Smith, Douglas B., and Streyle, Dale G., *An Inexpensive Real-Time Interactive Three-Dimensional Flight Simulation System*, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1987.
6. Oliver, Michael R., and Stahl, David J., *Interactive, Networked, Moving Platform Simulators*, M.S. Thesis, Naval Postgraduate School, Monterey, California, December 1987.
7. Silicon Graphics Inc., *IRIS User's Guide, MEX Window Manager*, Mountain View, California, 1987.
8. Silicon Graphics Inc., *4Sight User's Guide*, v. 1, Mountain View, California, 1988.
9. Hearn, Donald and Baker, M. Pauline, *Computer Graphics*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1986.
10. Silicon Graphics Inc., *IRIS User's Guide*, v. 1, Mountain View, California, 1987.
11. McConkle, Corinne and Nelson, Andrew H., *A Prototype Simulation System for Combat Vehicle Coordination and Motion Visualization*, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1988.
12. Silicon Graphics Inc., *IRIS GTX: A Technical Report*, Mountain View, California, 1988.
13. Silicon Graphics Inc., *Power Series A Family Overview*, Mountain View, California, September 1988.

Distribution List for Dr. Michael J. Zyda

Defense Technical Information Center,
Cameron Station,
Alexandria, VA 22314 — 2 copies

Library, Code 0142
Naval Postgraduate School,
Monterey, CA 93943 — 2 copies

Center for Naval Analyses,
4401 Ford Avenue
Alexandria, VA 22302-0268 — 1 copy

Director of Research Administration,
Code 012,
Naval Postgraduate School,
Monterey, CA 93943 — 1 copy

Dr. Michael J. Zyda
Naval Postgraduate School,
Code 52, Dept. of Computer Science
Monterey, California 93943-5100 — 200 copies

Mr. Bill West,
HQ, USACDEC,
Attention: ATEC-D,
Fort Ord, California 93941 — 1 copy

John Maynard,
Naval Ocean Systems Center,
Code 402,
San Diego, California 92152 — 1 copy

Duane Gomez,
Naval Ocean Systems Center,
Code 433,
San Diego, California 92152 — 1 copy

James R. Louder,
Naval Underwater Systems Center,
Combat Control Systems Department,
Building 1171/1,
Newport, Rhode Island 02841 — 1 copy